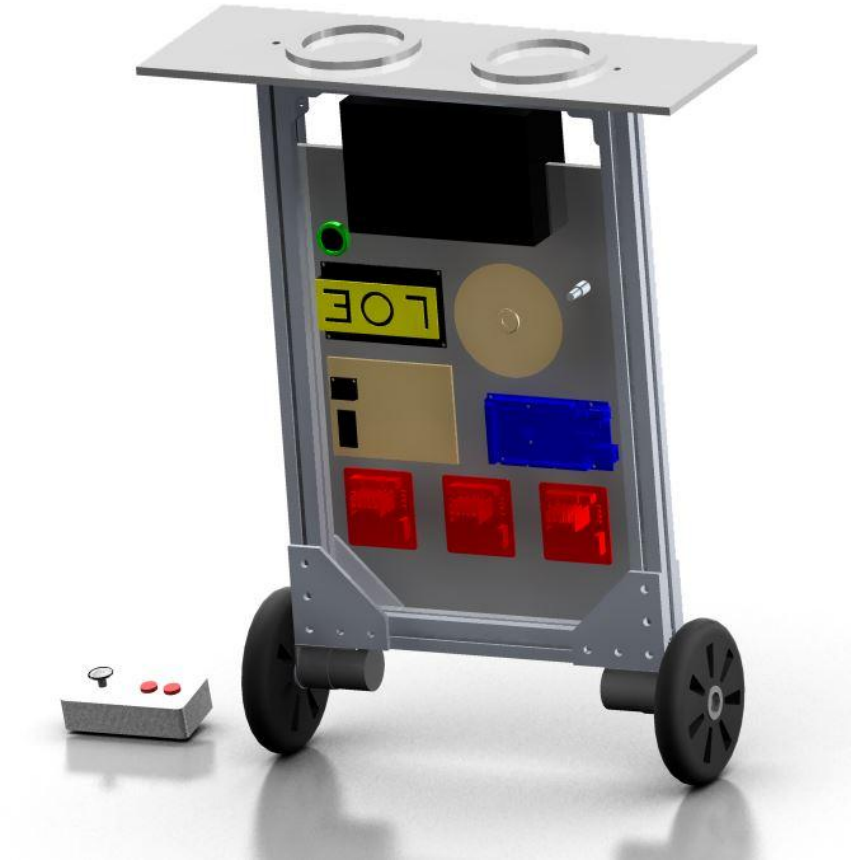
# DDV-IP



Group #12

Michael Habel

Robert Harvey

Tyler Reedy

Doug Swift

May 9, 2014

# Table of Contents

## Design Summary

The DDV-IP is an inverted pendulum driving machine. DDV-IP stands for Drink Delivery Vehicle – Inverted Pendulum. The concept is a two wheeled balancing robot that can deliver cold beverages to thirsty folks on hot summer days. All of the features of the DDV-IP result in an effective delivery vehicle while providing entertainment to the user. Figure 1 is an illustration of the DDV-IP. A wireless remote enables control of the device beyond the act of self-balancing.

**Figure 1** CAD rendered model of the DDV-IP with remote control.

To describe how the DDV-IP operates, a side view of the device is pictured in Figure 2. The concept is similar to balancing a broom on your fingertip. The pivot point of the device is its center of gravity, located near the top center. As the center of gravity departs from perfectly vertical, it begins to accelerate downward toward the ground. The IMU consists of accelerometers and gyroscopes that allow the angle of the center of gravity and the rate at which it is accelerating toward the ground to be calculated. This information is sent to a PID controller which determines the correction speed and direction required to set the pulse width modulated motors to in order to return the robot

to the neutral position. A trimming potentiometer used to adjust the desired neutral angle upon startup. The neutral angle set point, as well as the current angle can be monitored via the LCD display. This allows for the device to be fine-tuned for a given location as the unit is sensitive to changes in floor surfaces and ambient temperatures.

As a safety feature, when the DDV-IP is turned on, the motors will not power up until the unit is raised to the vertical position. At this time, a warning tone sounds to let the user know that the DDV-IP is fully activated and the motors will begin to spin.

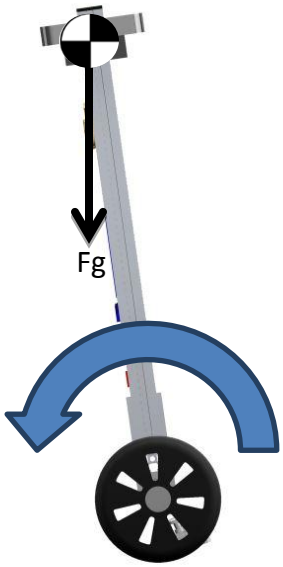



**Figure 2** DDV-IP operation.

The remote control operates wirelessly via a 2.4 GHz radio module, performing two axis control of the DDV-IP in addition to lighting control. The forward and backward motion of the DDV-IP is controlled via the y-axis of the joystick, while turning is controlled by the x-axis of the joystick. The forward and backward motion is created by proportionally adjusting the neutral angle with the amount of y-axis joystick travel. Button 1 on the remote control turns the entertaining PIC controlled light show on and off. The light show consists of a patterned routine of super bright LEDs mounted in the cup holders. Button 2 is a design consideration that allows for future implementation of additional functionality. The contents of the remote control can be seen in Figure 4.

## System Details

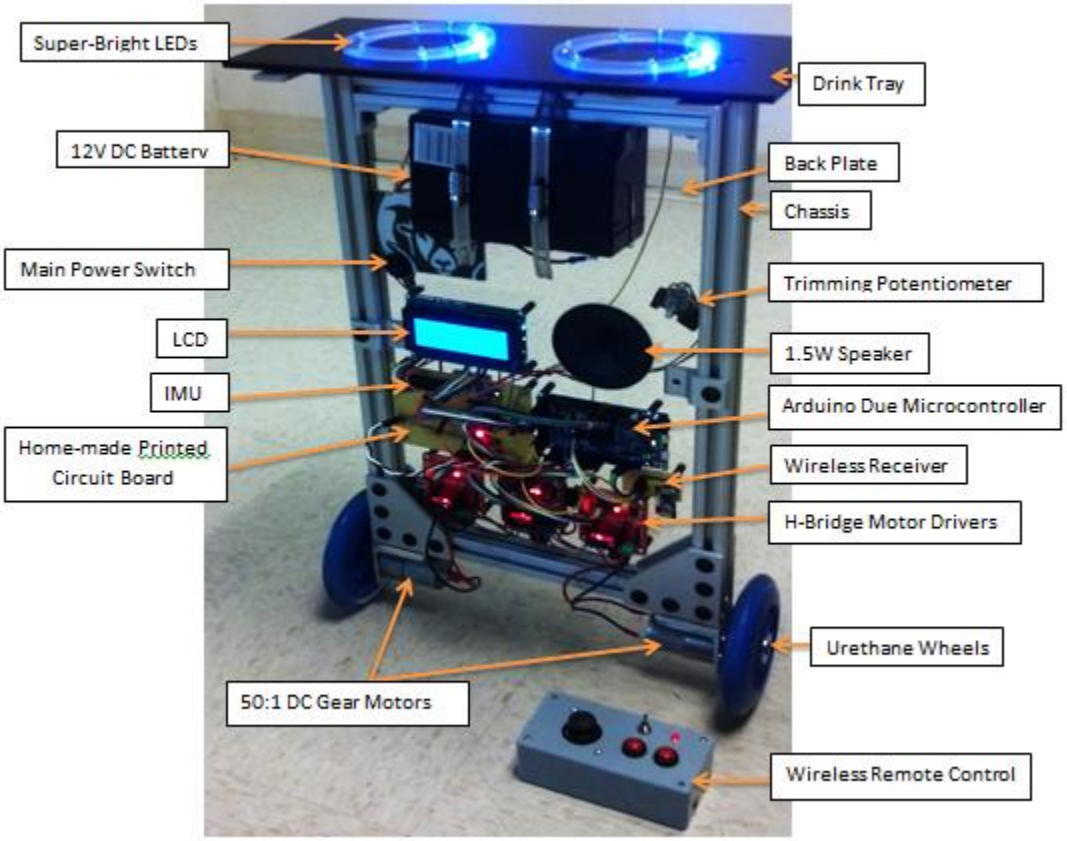



**Figure 3** DDV-IP configuration with labels.

Chassis: The chassis of the device consists of an extruded aluminum frame with an acrylic backing plate as seen in Figure 3. All parts required for device operation are mounted to the acrylic backing plate, with the exception of the battery and gear motors.

Propulsion System: The device is propelled by two independently controlled urethane wheels. The ability to stay upright and move is the result of using pulse-width-modulated, reversible gear motors. The gear motors are 50:1 DC motors. The motors are provided with power by three H-Bridge motor drivers which allow for bi-directional operation of the motors. The wheels, motors and H-bridges are pictured in Figure 3.

Control System: Balancing a two wheeled robot is a complex task. In order to balance, the robot must perform the following functions: retrieve data from the inertial motion unit (IMU), process this data using a PID controller implemented on the Arduino Due, and control the motors accordingly so that the robot maintains balance, even under slight perturbations. The IMU and Arduino Due can be seen in Figure 3. The heart of the DDV-IP operation lies in its use of the closed-loop feedback PID controller. The data

obtained from the IMU is used to determine necessary corrective and predictive motor actions, allowing the unit to stand upright. The unit's angle is determined from an arc-tangent operation of the accelerometers x-axis and y-axis data, the angle and change in angle are compared to the set-point, and the control system acts to correct any deviation from the set-point. The wireless remote control unit of the DDV-IP contains an Arduino Uno, which is used to send data to the unit's main microcontroller. These devices communicate via SPI protocol. Additionally, the Arduino Due controls the PIC16F88 which is mounted beneath the drink tray pictured in Figure 3. The PIC16F88 controls the entertaining light show, using logic to determine when to operate the LEDs.

Human Interface: The DDV-IP uses several forms of manual data input successfully. The remote control operates wirelessly via a 2.4 GHz radio module, performing two axis control of the DDV-IP. The forward and backward motion of the DDV-IP is controlled via the y-axis of the joystick, while turning is controlled by the x-axis of the joystick. The forward and backward motion is created by proportionally adjusting the neutral angle with the amount of y-axis joystick travel, thereby initiating forward or backward motion. Turning is initiated by superposing turn control data on balance output. This is accomplished by increasing the duty cycle of one motor and decreasing the other. Button 1 on the remote control toggles the entertaining PIC controlled light show. The light show consists of a patterned routine of super bright LEDs mounted in the cup holders. Button 2 is a design consideration that allows for future implementation of additional functionality. The contents of the remote control can be seen in Figure 4.

In addition to the remote inputs, the DDV-IP utilizes two potentiometers, one which allows the user to trim the set point angle of the unit during startup, and the other allows the user to adjust the contrast of the LCD screen. Both the main unit and the remote control utilize on-off toggle switches. Additionally, our device was able to communicate with a serial COMM port to experiment with tuning constants for the PID controller in real time.

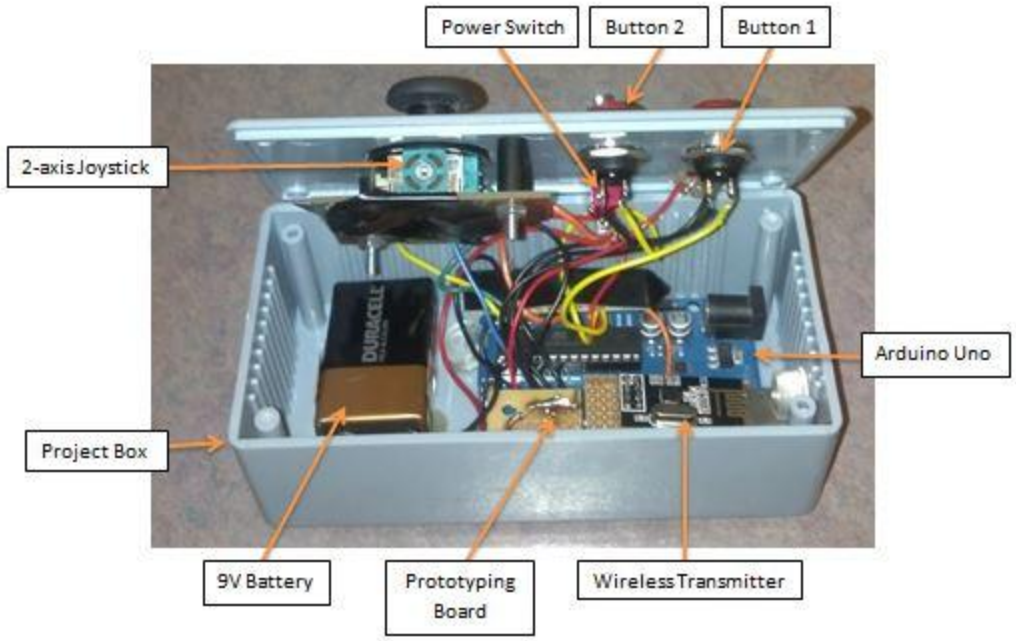



**Figure 4** Wireless remote control configuration with labels.

Output Display: The DDV-IP utilizes several output displays controlled by our main microcontroller. A 4x20 white-on-blue LCD displays several parameters important to the function of the machine. First, the LCD displays the actual angle at which the IMU is currently oriented with respect to gravity. The LCD also displays the current set-point angle which is used to compute error for the PID control system. The entertaining light show is controlled by a single 16F88 PIC microcontroller, which is enabled through the Arduino Due. This light show can be turned on and off by the user of the machine via Button 1 on the remote control unit. A pulse width modulated LED on the remote control indicates that power is being supplied to the remote control unit. The flashing "heartbeat" of the LED notifies the user that the Arduino Uno in the remote control unit is running through its code routines properly.

Audio Output: The audio output of the DDV-IP is controlled by the Arduino Due. As a safety feature, when the DDV-IP is turned on, the motors will not spin until the unit is raised to the vertical position. At this time, a warning tone sounds to let the user know that the DDV-IP is fully activated and the motors will begin to spin. The sound is produced by supplying a 500 Hz frequency to a transistor, resulting in sound from a 1.5 watt speaker. Initially, we were planning on using a mono audio amp, but due to electrical interference were forced to seek an alternative means of producing sound. Further discussion of this problem can be found in the *Lesson Learned* section.

Sensors: Automatic sensors and the use of the data they provide are at the heart of the operation. The DDV-IP uses one chip (the IMU) for the acquisition of inertial data. This chip gathers acceleration and angular rate data which is sent to the main microcontroller. The gyroscope measures the angular velocity of the device, while the accelerometers measure acceleration which is used to the angle with respect to gravity.

## Circuit Schematics/Wiring Diagrams

The main board circuit schematic, pictured in Figure 5, shows the pin connections from the Arduino Due to the IMU, LCD, speaker, wireless module, and H-bridge motor drivers.

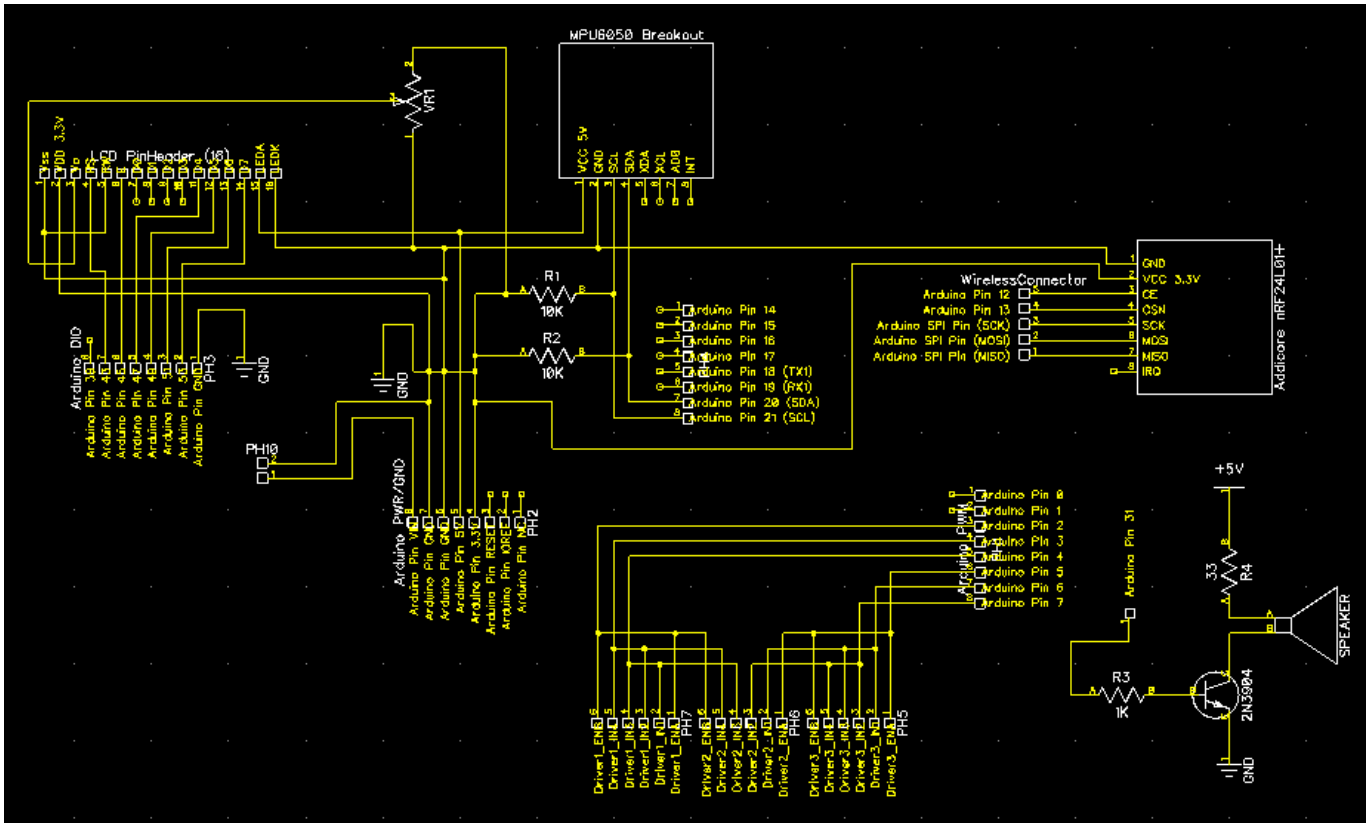



**Figure 5** Main board circuit schematic/wiring diagram.

The power board circuit schematic, pictured in Figure 6, shows how the 12V power supply is regulated to 9V and 5V for lower voltage power hubs.

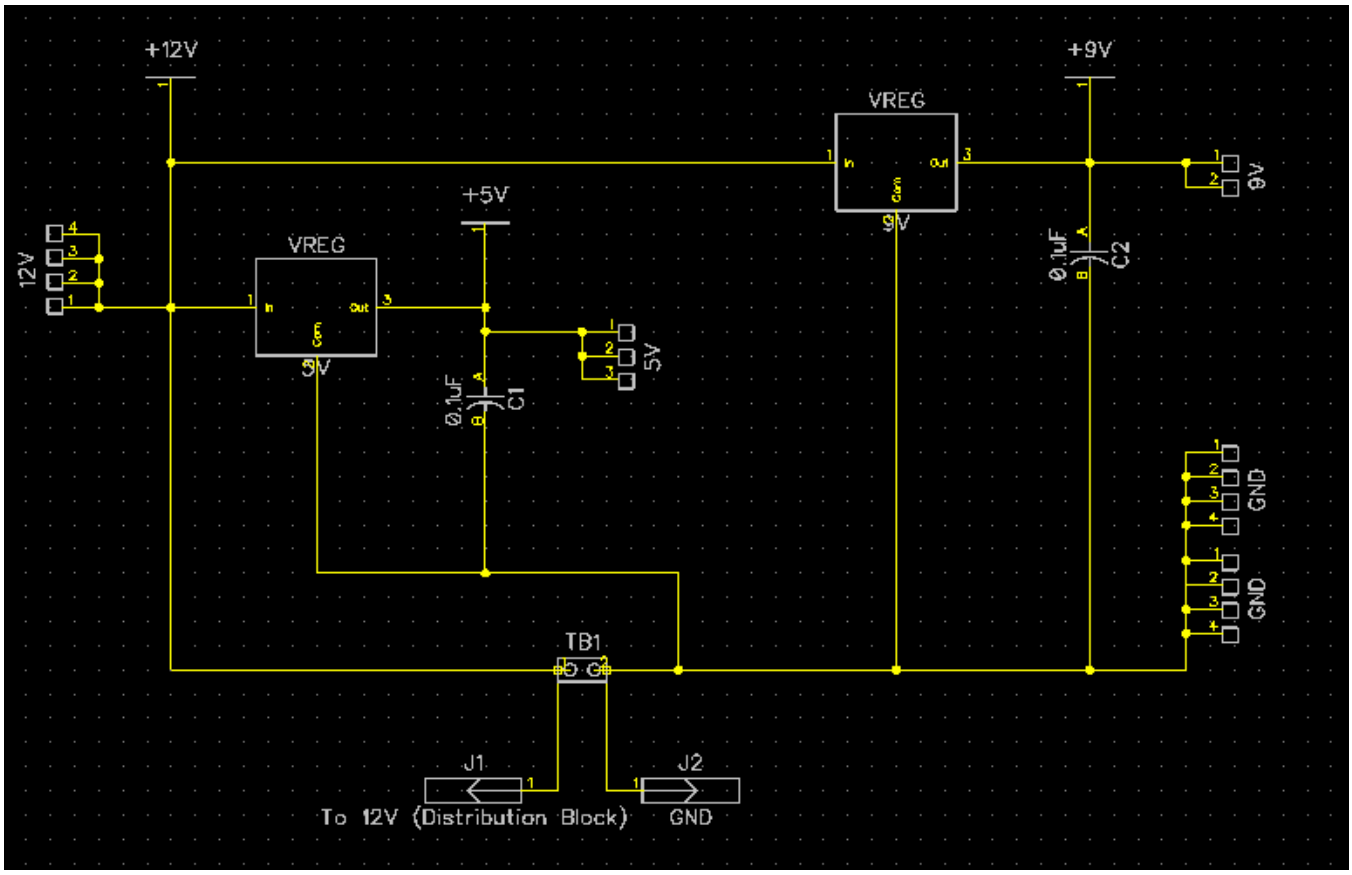



**Figure 6** Power board circuit schematic/wiring diagram.

The remote control circuit schematic, pictured in Figure 7, shows the pin connections from the Arduino Uno to the remote control components including Button 1, Button 2, the joystick, the power switch, the 9V power supply, the power LED, and the wireless module.

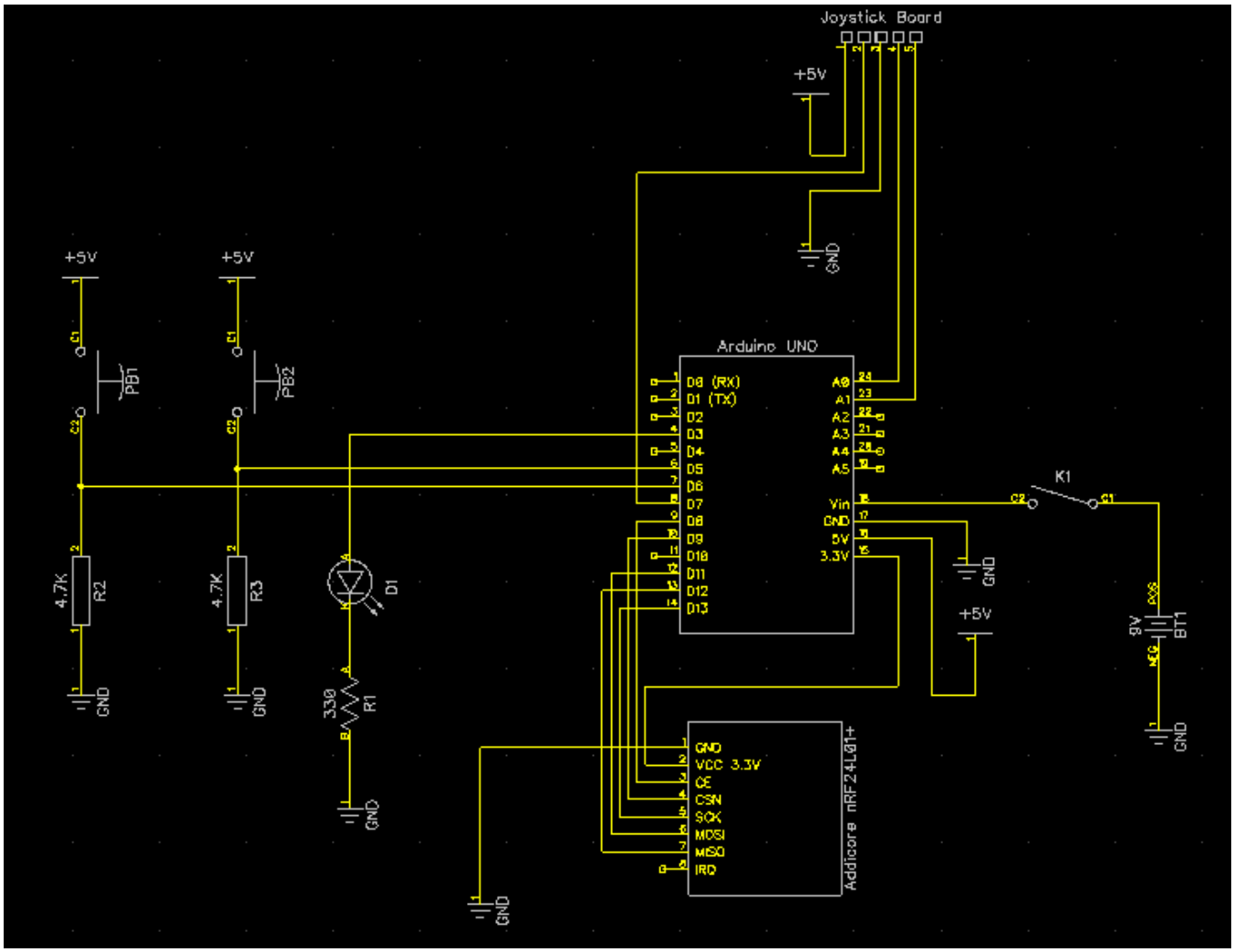



**Figure 7** Circuit schematic/wiring diagram of the remote control.

The DDV-IP power source is a 12 volt lead acid battery. 12 volts is the optimal voltage to drive each of the gear motors via the H-bridge motor drivers. Power distribution wiring can be seen in Figure 8. For ease of connections, a bus bar was inserted as a power distribution hub.

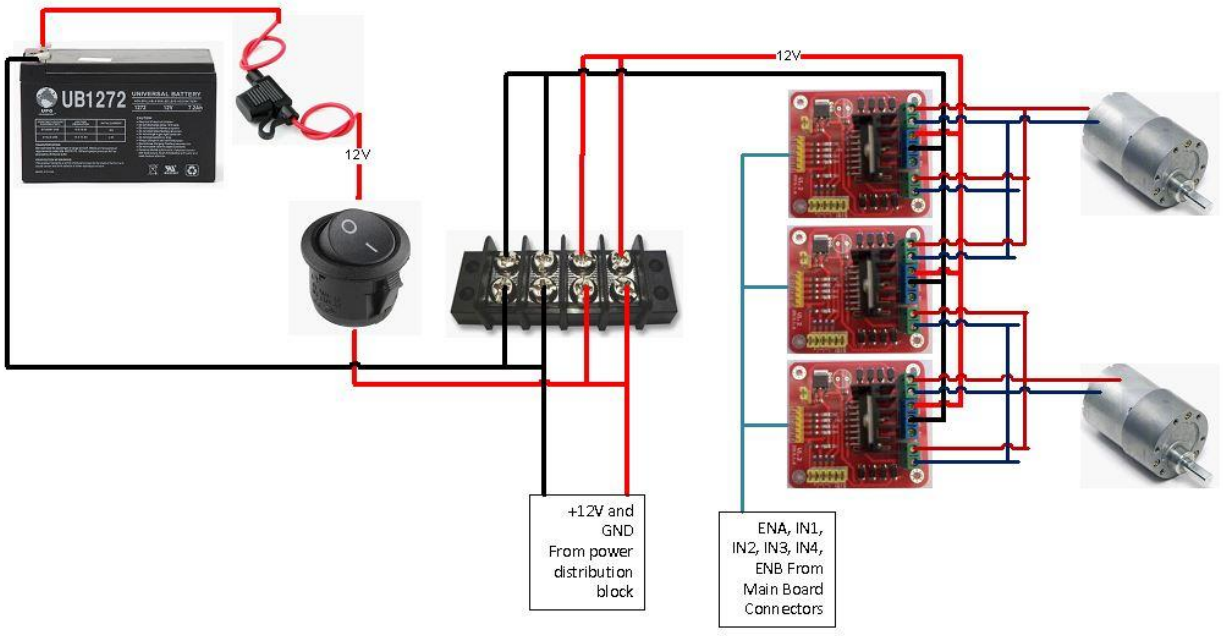



**Figure 8** Power drive system circuit schematic/wiring diagram.

The LED tray circuit schematic shown in Figure 9 displays the PIC16F88 used to control the entertaining light show. The PIC is powered by 5V and receives a digital signal from the Arduino Due to allow the PIC to turn the attached LEDs on or off. All resistors in the schematic are 100 ohm resistors and all LEDs are RGB LEDs, wired only to the blue diode.

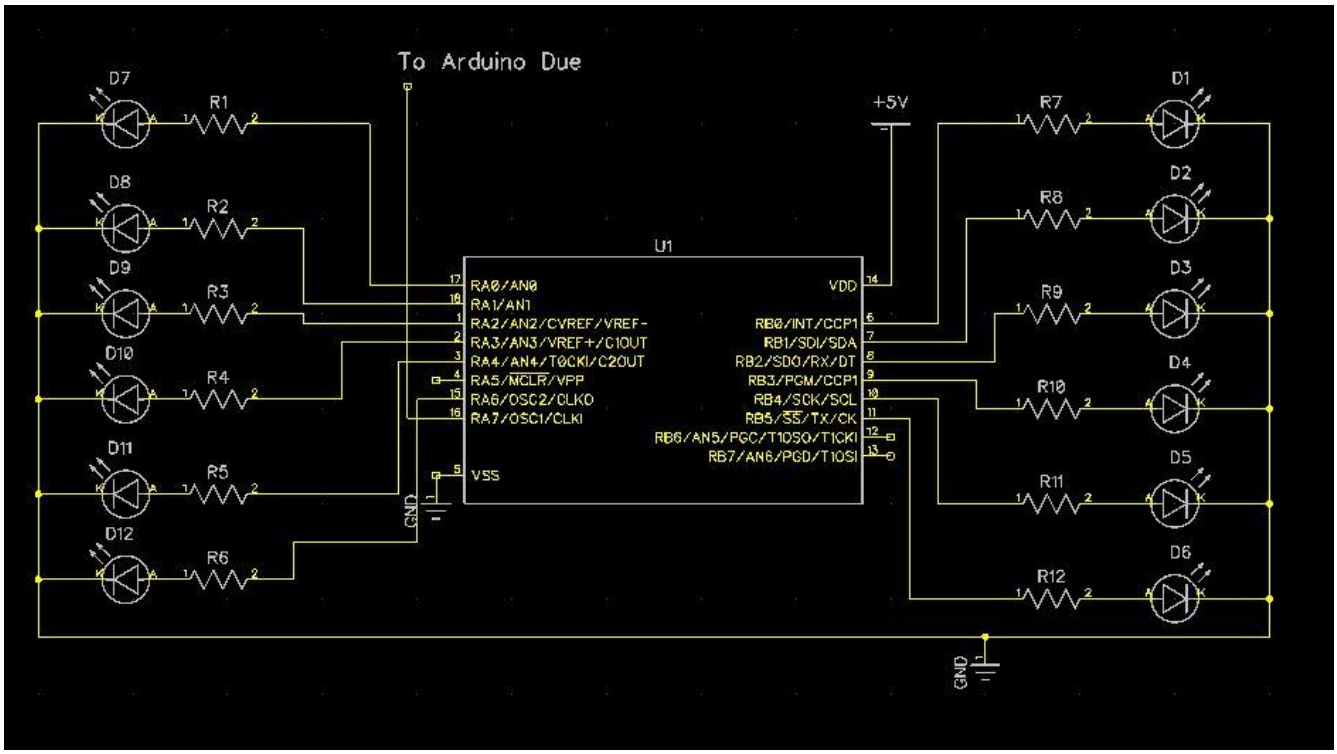



**Figure 9** LED Tray circuit schematic/wiring diagram.

## Functional Diagrams

Main Functional Diagram:

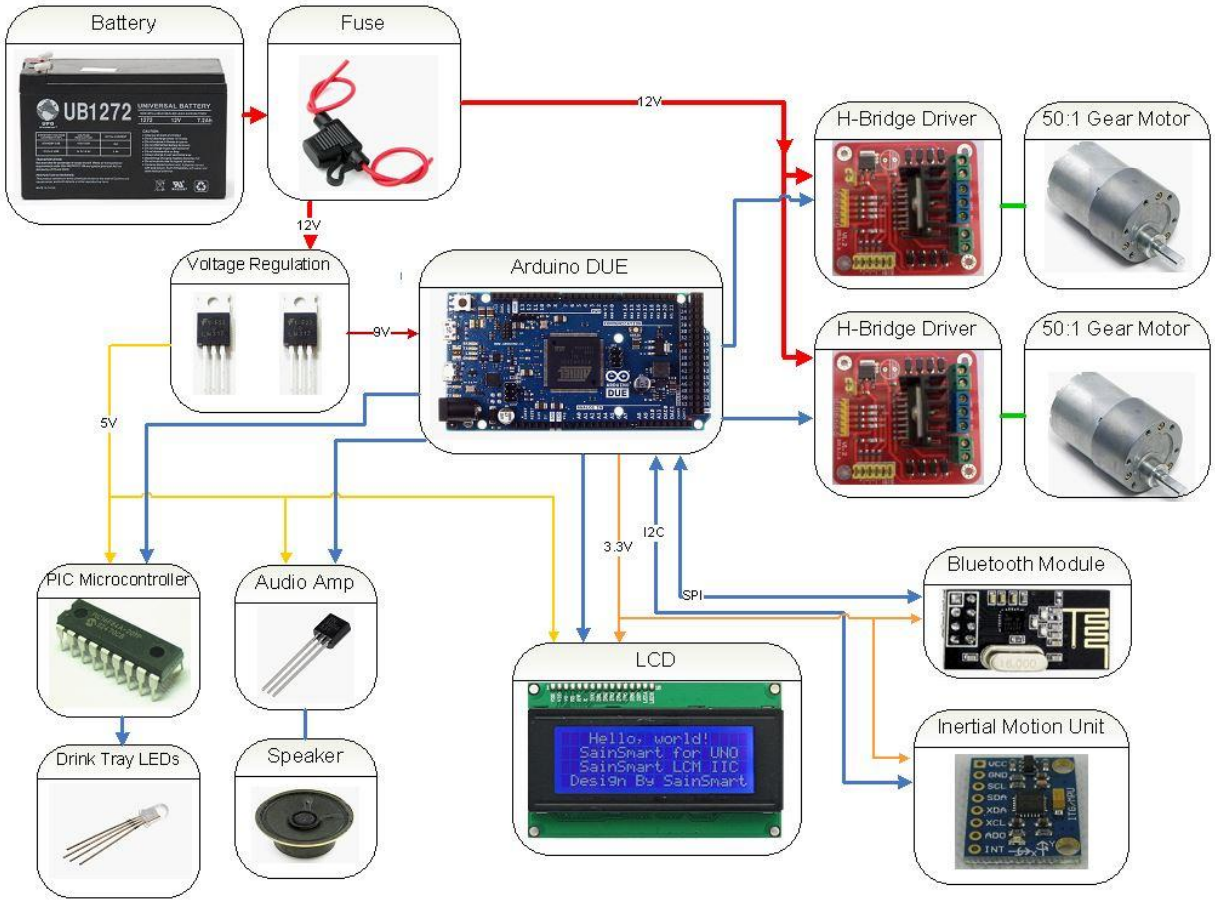



**Figure 10** Main functional diagram.

Remote Functional Diagram:

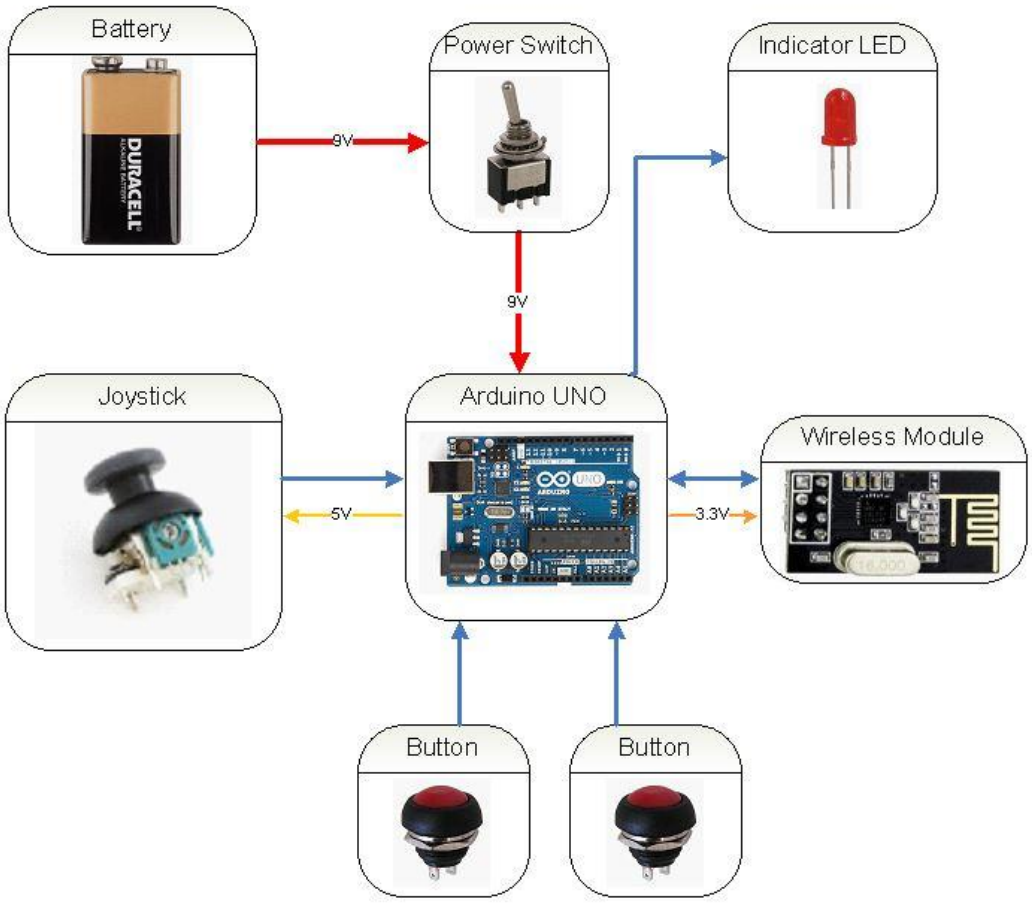



**Figure 11** Remote control functional diagram.

## Flow Charts

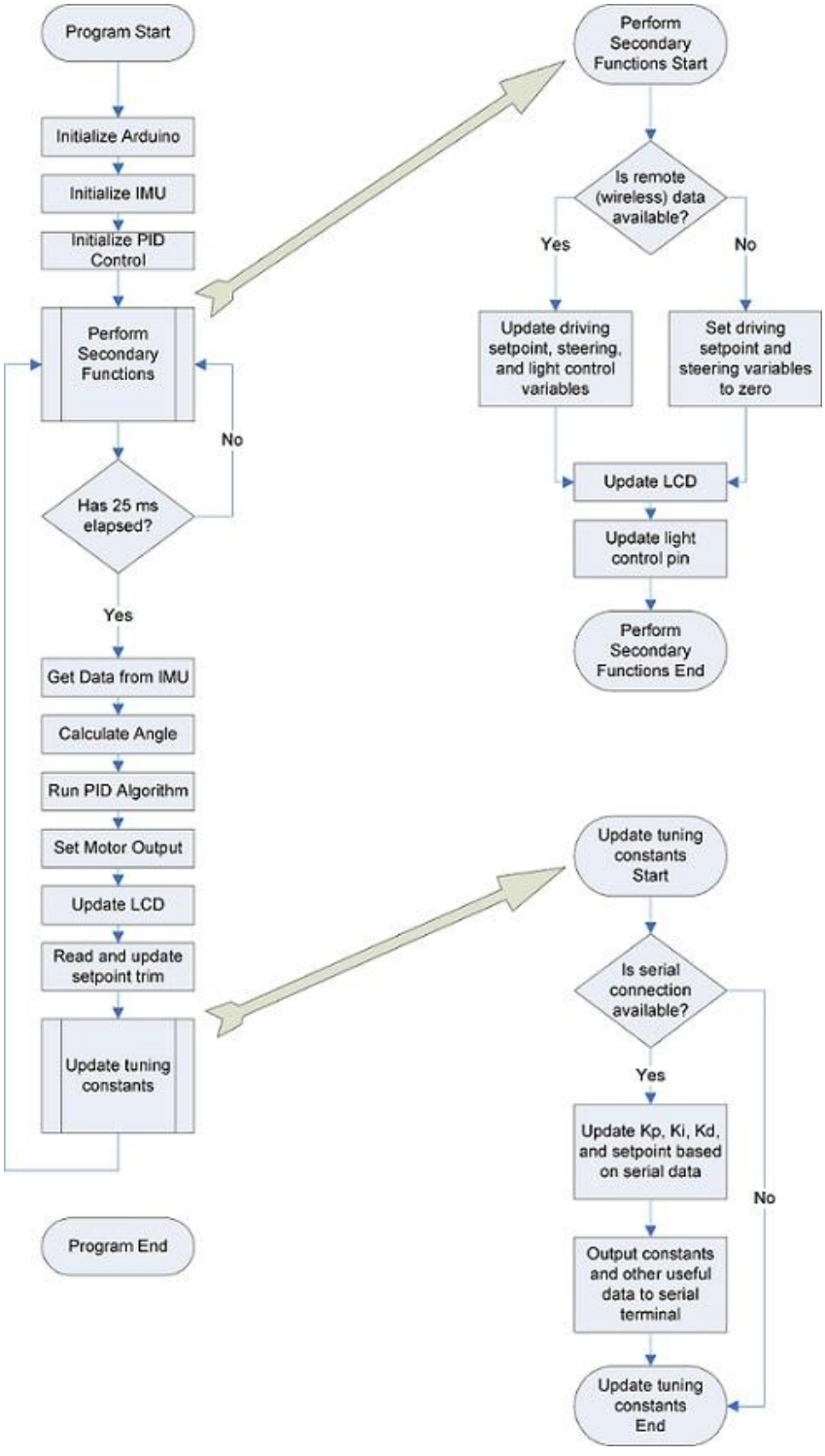



**Figure 12** Main software flowchart.

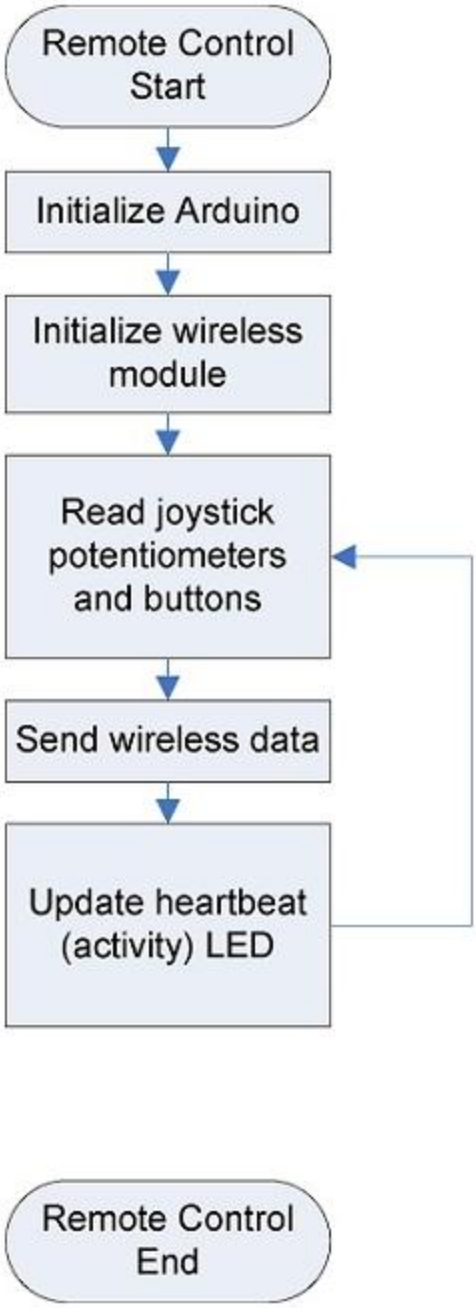



**Figure 13** Remote control flowchart.

## Design Evaluation

Overall, our project has met and exceeded the general categorical requirements set forth by the project assignment. The DDV-IP contains more than one element in each of the six functional element categories and the success of each of these elements is described.

### Output Display

- An LCD provides useful information to the user
- LEDs
    - LEDs driven by a PIC16F88 microcontroller provide a light show on the drink tray.
    - An LED on the remote control indicates power being provided to the remote unit and successful operation of the Arduino inside the remote control unit.

All the output display functions worked properly, were repeatable and were a necessary and integral part of our device. Use of this type of LCD was fairly straight forward, and required wiring and software manipulation, but it was not covered in class. The LEDs and their use were covered briefly in lab.

### Audio Output Device

- Higher volume using a transistor assisted amplified signal.
- A speaker is used to play software-generated sound effects.

Our original sound output device was to be a mono audio amp, but due to electromagnetic interference, we were forced to use transistor assisted amplification instead. The generation of sound through a transistor amplifier functioned as designed, was repeatable, and was integral to the project. Additionally, the use of a transistor to amplify a software generated audio signal was not covered in class or lab.

### Manual Data Input

- A two-axis joystick along with a 2.4 GHz RF wireless link allows a user to drive the device.
- Two potentiometers are used to trim the neutral angle and adjust the LCD contrast, respectfully.
- Two buttons along with the 2.4 GHz RF wireless link allow the user to control various functions on the device.
- Two power switches allow the user to disconnect power from the device and the remote control, respectfully.

All of the DDV-IP's manual inputs worked as designed, and were repeatable. The use of a two axis joystick and associated analog-to-digital converters were not covered in class

or in lab, though their use was not difficult, as it required reading voltage values from two potentiometers and converting their position to a digital number. The use of the wireless modules used to transmit the data from the remote required extensive research and programming due to the complexity of the modules and lack of preexisting libraries for the Arduino DUE. Use of the external serial computer COMM port was not discussed in class, lab or the book.

**Automatic Sensor**

- A 3-axis gyroscope senses changes in angular rotation of the IMU and sends data to the main microcontroller.
- A 3- axis accelerometer senses acceleration with respect to gravity and sends data to the main microcontroller.

The IMU on the DDV-IP was an integral part of the device and its reliability and the repeatability of its use allowed this project to happen. The use of gyroscopes was not covered in class, but the use of accelerometers was covered briefly, though we had been using the accelerometers on the device for several weeks before covering the material.

**Actuators, Mechanisms, and Hardware**

- Two PWM, speed-controlled, reversible gear motors, powered through paralleled H-bridge motor drivers deliver drive power to the devices, based on PID output signals.

To ensure that the device would stay upright, reversible motors capable of variable speeds were required. The use of H-bridge motor drivers was not discussed in class, though the book does cover their use. The use of a PWM signal was taught in lab.

**Logic, Processing, and Control**

- A Closed-loop feedback control scheme (Motion data, PID, and motor control) was used to control the motor output of the device and keep it upright.
- Multiple interfaced micro controllers (Due, Uno and PIC) were used to control the wireless transmission and reception of data, and to control the entertaining light show.
- Several calculations were used to determine the angle of the device and to control the motor outputs. Numerous variables were used which used data storage and retrieval routines.
- The light show controlled by the PIC used programmed logic to determine the appropriate time to provide the show.

The use of the PID controller required extensive research to implement and required a great amount of tuning in order to achieve the level of stability that was achieved. Although it was

not perfect, it was repeatable, and could reliably keep the device upright in most normal driving situations.

**Miscellaneous**

- Homemade printed circuit boards cleaned up the routing of wires and cables.
- Custom cables and wires allowed for neat routing of necessary equipment.
- T-slotted aluminum was used for the frame to ensure durability.

The design elements that were introduced for this project were not covered in class, and are useful ways to help clean a project up and keep the "Rat's nests" from forming, which can hinder development efforts.

## Partial Parts List

**Table 1** Partial parts list.

| Qty | Model Number | Part Name | Price (each) | Price (total) | Vendor |
|---|---|---|---|---|---|
| | | **Chassis/Structural** | | | |
| 2 | custom1 | Motor Bracket - Custom Fabricated | $ 0.01 | $ 0.02 | Scrap Bin |
| 2 | custom2 | Motor Bracket Riser - Custom Fabricated | $ 0.01 | $ 0.02 | Scrap Bin |
| 1 | custom3 | Back Plate, Acrylic, 17.25"x10.25"x0.25" | $ 10.00 | $ 10.00 | Fort Collins Plastic |
| 1 | custom4 | LED Top Tray, Acrylic. 17.25"x6"x0.25" | $ 8.00 | $ 8.00 | Fort Collins Plastic |
| | | | | | |
| | | **Main Unit Electronic Hardware** | | | |
| 1 | B00GB37EN6 | Microcontroller Board, Arduino Due | $ 46.95 | $ 46.95 | www.amazon.com |
| 2 | 1104 | 50:1 Metal Gear Motor | $ 24.95 | $ 49.90 | www.pololu.com |
| 1 | B008BOPN40 | Kootek Arduino GY-521 MPU-6050 Module | $ 6.00 | $ 6.00 | www.amazon.com |
| 1 | 5063-4513 | Speaker - 1.5W 8 ohm | $ 1.95 | $ 1.95 | www.sparkfun.com |
| 3 | L298 | Dual H-Bridge Motor Driver | $ 5.57 | $ 16.71 | www.amazon.com (nooElec) |
| 1 | B00E594ZX0 | 2 pcs nrf24lo1+2.4 Ghz wireless transciever | $ 6.98 | $ 6.98 | www.amazon.com |
| 3 | custom5 | Printed Circuit Board | $ 4.00 | $ 12.00 | Home-made |
| 1 | PIC16F88 | PIC Microcontroller | $ 0.01 | $ 0.01 | MicroChip free sample program |
| | | | | | |
| | | **Remote Controller** | | | |
| 1 | PB185 | Project Box | $ 5.49 | $ 5.49 | Mountain States Electronics |
| 1 | A000073 | Microcontroller Board, Arduino Uno | $ 24.95 | $ 24.95 | www.amazon.com |
| 1 | - | Joystick | $ 0.01 | $ 0.01 | Salvaged from video game controller |
| 2 | PB-179 | Push Button - NO | $ 1.00 | $ 2.00 | www.allelectronics.com |
| | | | | | |
| | | **Miscellaneous** | | | |
| 1 | | Miscellaneous/ Non-interesting Parts | $ 230.11 | $ 230.11 | Various Suppliers |
| | | | | | |
| | | **Total Build Cost** | | $ 421.10 | |

## Lessons Learned

1. Start early to avoid a last minute crunch.
    - Our group planned adequately by realizing the time commitment that our complicated project would take. We got an early start and had a general schematic including needed components weeks before the deliverable was due. This enabled us to work on other aspects such as wiring diagrams as we waited for parts to arrive.

2. Understanding interference issues between components.
    - Our project was acting up and locking up the Arduino for reasons that were unknown. After some tinkering, it was found that the motors and the 250 kHz class D pulse width modulated audio amplifier were introducing too much noise on the Arduino power line. Capacitors were placed across the motor leads to alleviate some of the problem. The same approach did not work for the audio amp however, so this problem was solved by utilizing a different approach in amplifying the sound. A transistor was used in place of the audio amplifier. The transistor allowed us to still output a frequency of 500 Hz while drastically reducing the noise and electrical interference. Without changing any code, the Arduino was back to functioning properly.

3. Wire organization not only makes a more aesthetically pleasing project but makes a more reliable product.
    - There are a lot of components in a relatively small space on our project. Our group decided from the beginning that we were out to make a durable and aesthetically pleasing project. With this in mind, considerable time was taken for wiring layout and design. We decided to create our own printed circuit boards to help in reducing the almost inevitable "rats nest" of wires. This way, we could create a central hub to run all the wires to and from. An added benefit of using a printed circuit board customized to our needs was that we could run traces such that an output block of neighboring wires could be run together to an input block of neighboring inputs on the Arduino. This greatly reduced the "rats nest" effect and increased organization.
    - Utilizing printed circuit boards enabled the use of soldered joints. Soldered joints are much more reliable than simply plugging wires into a breadboard. Wires will not unexpectedly fall out or accidentally short to something else.

4. Check microcontroller functions and compatibility with other hardware
    - The microcontroller is at the heart of our project. Being such a main component of our design, compatibility with our ideas was a necessity. After determining

what the project is to do, make sure that the microcontroller is able to handle all of the needs. Our group chose to go with an Arduino Due for our project for three reasons.

- The first reason was the faster, 84 MHz operating speed of the Due. Our inverted pendulum requires PID control to keep it balanced and therefore the ability to process information quickly. The high operating speed allowed for increased data acquisition from the gyroscope and accelerometer. A higher input speed allows for a more accurate representation of the motion of the unit.
- The Due is a 32 bit microcontroller and therefore handles floating point math more quickly than other microcontrollers. Fractions of an angle can affect the unit greatly and those fractions need to be carried through the PID calculations in order to optimize the correction and remain upright.
- The Due operates on 3.3 volts. Our sensors run off of 3.3 volts as well, so this helped avoid the use of logic level convertors.

- The Due is a newer microcontroller. Being a newer unit, not as many people have worked with it like the Arduino Uno or Nano. A large problem that we ran into was that not all libraries are compatible with the Due. We used a PID library, LCD library, inertial motion unit (IMU) library, wire library, $I^2C$dev library, MPU650 library, math library, SPI library, RF24 wireless library and the standard int (data size) library. We found some libraries worked with no modification but others required slight or significant modifications. The largest trouble came when trying to set up the wireless module. There was no library written specifically for the Due; the only libraries that could be found were for the Uno and Nano. Large modifications were made to one such library to make it compatible with the Due. This required a large amount of time and frustration to properly configure. That was the only downside we found in using the newer technology of the Due.

5. Keep things simple initially
    - Many extravagant ideas were introduced throughout the course of the project. These superfluous ideas were in addition to the initial ideas meeting the design requirements. Countless times throughout the project build we had to remind ourselves to stick to the original plan and meet the design requirements first. Only after the design requirements were met, did we allow ourselves to introduce additional design concepts to add to the function and aesthetics.

# Appendix

## Main Program Code:

```
/*********************************************************************************

define statements

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

#define LED_PIN 13       //onboard LED
#define MOTOR1_ENABLE 2   //Motor driver pins
#define MOTOR1_IN1 4
#define MOTOR1_IN2 3
#define MOTOR2_ENABLE 5
#define MOTOR2_IN1 6
#define MOTOR2_IN2 7
#define SPEAKEROUT 31      //speaker output pin
#define comp_filter_const 0.992    //complimentary filter constant (0<=const<=1)
#define DT 25    //sample time (ms)
#define motorStart 25      //motor PWM functions will receive this as min value
#define LeftMotorTrim 0
#define RightMotorTrim 1
#define CE_PIN   12      //wireless module CE pin
#define CSN_PIN 13      //wireless module CSN pin
#define angleAverageNum 1
#define remoteYaxisNatural 115    //resting state of Y (vertical) axis on remote control.
#define remoteXaxisNatural 122    //resting state of X (Horizontal) axis on remote control.
#define TrayLEDPin 28


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

Libraries

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

#include "Wire.h"  // Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE
implementation is used in I2Cdev.h
#include "I2Cdev.h"  // I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h
files for both classes must be in the include path of your project
#include "MPU6050.h"
#include "Math.h"
#include <LiquidCrystal.h>
#include <SPI.h>
//#include <nRF24L01.h>
#include <RF24.h>
#include <stdint.h>
```

```
LiquidCrystal lcd(43, 45, 47, 49, 51, 53);  //(RS, En, D4, D5, D6, D7)    // initialize the library
with the numbers of the interface pins
#include <PID_v1.h>
/*-----( Declare objects )-----*/
MPU6050 accelgyro;    //define MPU6050 as accelgyro object
RF24 radio(CE_PIN, CSN_PIN); // Create a Radio object
/*-----( Declare Variables )-----*/
const uint64_t pipe = 0xE8E8F0F0E1LL; // Radio: Define the transmit pipe
uint8_t  dataBuffer[32];
int lastData[2] = {0,0};
float gyro_scale = 131.0;  //value = [(2^16)-1]/(gyro scale (degrees/second))
float accelerometer_scale = 50.0;    //scale value for raw accelerometer data from IMU
float idealAngle= 93.97;
float myKp = 154.0;  //154.0;    values that work reasonably well on carpet
float myKi = 105.0;  //225.0;
float myKd = 0.90;  //0.00;
float Setpoint, Input, Output;    //PID variables
PID BalancePID(&Input, &Output, &Setpoint,myKi, myKp, myKd, DIRECT);    //Specify the
links and initial tuning parameters
float currentAngle, accelAngle, gyroAngle, gyroRateY;  //angles and gyro data
float lastAngle = idealAngle;
float angleArray[angleAverageNum];
int avgAngleIndex = 0;     //create index for average angle array at beginning of array
int LeftMotorDrive = 0;//128;
int RightMotorDrive = 0;//-128;
int beginningTime, endTime, loopTime, computeBeginTime, computeLastTime;    //debug
variables for timing
bool blinkState = false;    //blink state variable (Debug, but useful for program running indicator)
bool TrayLEDState = 1;
unsigned int loopCount = 0;    //used so remote vals are only read every <x> control loops
signed int remote_yaxisSetpoint;
/**********************************************************************************

Initialization

**********************************************************************************

void setup() {
   Serial.begin(115200);
   Wire.begin();  // join I2C bus (I2Cdev library doesn't do this automatically)
   pinMode(LED_PIN, OUTPUT);        // configure Arduino LED alive blinker
   //Assign Motor Output Pins, and set them Low (off):
   pinMode(MOTOR1_ENABLE, OUTPUT);
   pinMode(MOTOR1_IN1, OUTPUT);
   pinMode(MOTOR1_IN2, OUTPUT);
   pinMode(MOTOR2_ENABLE, OUTPUT);
   pinMode(MOTOR2_IN1, OUTPUT);
```

```
  pinMode(MOTOR2_IN2, OUTPUT);
  digitalWrite(MOTOR1_ENABLE, LOW);
  digitalWrite(MOTOR1_IN1, LOW);
  digitalWrite(MOTOR1_IN2, LOW);
  digitalWrite(MOTOR2_ENABLE, LOW);
  digitalWrite(MOTOR2_IN1, LOW);
  digitalWrite(MOTOR2_IN2, LOW);
  //Assign Speaker Output Pin and set it Low (off):
  pinMode(SPEAKEROUT, OUTPUT);
  digitalWrite(SPEAKEROUT, LOW);
  //Assign Tray Led Signal Pin and set it High (on):
  pinMode(TrayLEDPin, OUTPUT);
  digitalWrite(TrayLEDPin, LOW);
  delay(500);  //delay to allow power to reach a stable state
  // set up the LCD's number of columns and rows, and set curson to beginning:
  lcd.begin(20, 4);
  lcd.setCursor(0, 0);
  lcd.print("Angle: ");
  accelgyro.initialize();       // initialize MPU-6050
//   delay(500);  //delay to allow IMU to reach a stable state
  accelgyro.setFullScaleAccelRange(MPU6050_ACCEL_FS_8);    //set accelerometer scale 8G
  accelgyro.setDLPFMode(2);    //set the Digital LPF to: Accel - 94 Hz, Gyro - 98 Hz
  radio.begin();
  radio.openReadingPipe(1,pipe);
  radio.startListening();
  getMotionData();    //get initial data for variables
  getCurrentAngle(accelAngle);      //added to get some values initialized
  currentAngle = accelAngle;
//   getCurrentAngle();
  BalancePID.SetSampleTime(DT);
  BalancePID.SetOutputLimits(-255+motorStart,255-motorStart);
  //initialize the variables we're linked to
  Input = accelAngle;
  Setpoint = idealAngle;
  while(abs(accelAngle - idealAngle) > 1){
    getMotionData();      }
  currentAngle = accelAngle;
  SpeakerBeep();  //500Hz for 500ms
  BalancePID.SetMode(AUTOMATIC);  //turn on PID
  computeLastTime = millis();    //initialize computLastTime var for PID timing
 }
/*****************************************************************************

Main Program

*****************************************************************************/
```

```
void loop() {
       if(loopCount == 5){
         loopCount = 0;
         if(checkRadio(dataBuffer)){
//           for(int i=0; i<sizeof(dataBuffer)-5; i++){
//           Serial.print(i); Serial.print(": "); Serial.print(dataBuffer[i]); Serial.print("  ");
//           }
//         Serial.println();
         }
       }
         if(millis() - computeLastTime >= (DT)){
           loopCount++;
           computeLastTime = millis();
           getMotionData();
           getCurrentAngle(UpdateAverageAngle(accelAngle));
           Input =  currentAngle; //UpdateAverageAngle(currentAngle);  //currentAngle; //
           BalancePID.Compute();
           MotorLogic(Output);
           lcd.setCursor(7, 0);        //display current angle
           lcd.print(currentAngle);    // ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
           getSerialConstant();
           UpdateTrimAndRemote();
         }
//   endTime = micros();
 //   loopTime = endTime-beginningTime;
//    Serial.println(endTime-beginningTime);
//    delay(100);
    // blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
    delay(2);
}
/*********************************************************************************

Functions

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

void UpdateTrimAndRemote(){
  signed int potVal, remote_yaxis, remote_xaxis;
       potVal = analogRead(A0);
       potVal -= 512;
       potVal /= 6;
       if(dataBuffer[0] != 0){
       remote_yaxis = ((dataBuffer[0]+lastData[0])/2-remoteYaxisNatural);    //average two values
of remote yaxis data and scale to -127<x<128
       lastData[0] = dataBuffer[0];    //maintain value used in average
```

```
        if(remote_yaxisSetpoint < remote_yaxis){
         remote_yaxisSetpoint += (remote_yaxis - remote_yaxisSetpoint)/2;
        }
        else if(remote_yaxisSetpoint > remote_yaxis){
         remote_yaxisSetpoint -= (remote_yaxisSetpoint - remote_yaxis)/2;
        }
      remote_xaxis = ((dataBuffer[1]+lastData[1])/2-remoteXaxisNatural)/2;    //average two
values of remote xaxis data and scale to (-127<x<128)/2
      lastData[1] = dataBuffer[1];    //maintain value used in average
        LeftMotorDrive = remote_xaxis;
        RightMotorDrive = -remote_xaxis;
      }
      else{
       remote_yaxis = 0;
      LeftMotorDrive = 0;
       RightMotorDrive = 0;
      }
      Setpoint  = idealAngle + (float)potVal*0.01 + (float)remote_yaxisSetpoint*0.02;
      lcd.setCursor(7, 2);        //display current angle
      lcd.print(Setpoint);    // ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
      if(dataBuffer[3] == 1){
       ToggleTrayLED();
      }
}
//getMotionData requests data from the IMU and modifies the values in accelAngle and
gyroRateY
void getMotionData(){
     signed int ax, az;
     ax = 0; az = 0; gyroRateY = 0;
     int i;
   // read raw accel/gyro measurements from device:
   for (i=0; i<10; i++){
        ax += accelgyro.getAccelerationX();
        az += accelgyro.getAccelerationZ();
        gyroRateY += accelgyro.getRotationY();
   }
        ax = ax/(float)i;
        az = az/(float)i;
        gyroRateY = -gyroRateY/((float)gyro_scale*(float)i);
        ax/=(float)accelerometer_scale;
        az/=(float)accelerometer_scale;
        accelAngle = (float)(atan2(ax,az))*RAD_TO_DEG;    //calculate angle from accelerometer
data

}
void getCurrentAngle(float accel_Angle){
```

```
  currentAngle =
(float)(comp_filter_const)*(currentAngle+(gyroRateY*((float)DT/1000.0)))+(1-
comp_filter_const)*accel_Angle;
}
float UpdateAverageAngle(float newAngle){
  float averageAngle = 0;
  angleArray[avgAngleIndex] = newAngle;    //insert new angle into array
  if(avgAngleIndex == (angleAverageNum - 1)){  //increment pointer; wrap around if at end
    avgAngleIndex = 0;
  }
     else{
      avgAngleIndex++;    //move pointer
     }
    //compute average
    int j;
    for(j=0; j<angleAverageNum; j++){
      averageAngle += angleArray[j];
    }
    averageAngle = averageAngle/(float)j;
  return averageAngle;
}
void MotorLogic(int value){
  signed int LeftValue = value + LeftMotorDrive;
  signed int RightValue = value + RightMotorDrive;
 if((LeftValue) < 0){
      Motor1Control(0, (abs(LeftValue))+motorStart);
 }
 else if((LeftValue) >= 0) {
   Motor1Control(1, (LeftValue)+motorStart);
 }
  if(RightValue < 0){
      Motor2Control(0, (abs(RightValue))+motorStart);
 }
 else if(RightValue >= 0) {
   Motor2Control(1, (RightValue)+motorStart);
 }
}
//Motor Control: 0-->  backwards    1--> forwards
void Motor1Control(bool rot_direction, int rot_speed){
  digitalWrite(MOTOR1_ENABLE, LOW);
  if(rot_direction == 0){
   digitalWrite(MOTOR1_IN1, HIGH);
   digitalWrite(MOTOR1_IN2, LOW);
  }
  else {
   digitalWrite(MOTOR1_IN1, LOW);
```

```
   digitalWrite(MOTOR1_IN2, HIGH);
  }
  analogWrite(MOTOR1_ENABLE, rot_speed + LeftMotorTrim);
}
void Motor2Control(bool rot_direction, int rot_speed){
  digitalWrite(MOTOR2_ENABLE, LOW);

  if(rot_direction == 0){
   digitalWrite(MOTOR2_IN1, HIGH);
   digitalWrite(MOTOR2_IN2, LOW);
  }
  else {
   digitalWrite(MOTOR2_IN1, LOW);
   digitalWrite(MOTOR2_IN2, HIGH);
  }
  analogWrite(MOTOR2_ENABLE, rot_speed + RightMotorTrim);
}
bool checkRadio(uint8_t* array){
   if ( radio.available() )
  {
    // Read the data payload until we've received everything
    bool done = false;
    while (!done)
    {
      // Fetch the data payload
      done = radio.read(array, sizeof(array));
    }
    return 1;
  }
  else
  {
     return 0;
  }

}
//void SpeakerBeep(int frequency, int duration){          //takes frequency and duration as args and
plays a tone
//  float T = 1/frequency;  float durationCount = duration / T;
//  T /= 2;  //get half period
//  T *= 1000;  //multiply by 1000 for proper use in micros function.
//    for(int c = 0; c<durationCount; c++){
//       digitalWrite(SPEAKEROUT, HIGH);
//       delayMicroseconds(T);
//       digitalWrite(SPEAKEROUT, LOW);
//       delayMicroseconds(T);
//    }
```

```
//    digitalWrite(SPEAKEROUT, LOW);
//}
void SpeakerBeep(){          //takes frequency and duration as args and plays a tone
   for(int c = 0; c<350; c++){
      digitalWrite(SPEAKEROUT, HIGH);
      delayMicroseconds(1000);
      digitalWrite(SPEAKEROUT, LOW);
      delayMicroseconds(1000);
   }
   digitalWrite(SPEAKEROUT, LOW);
}

void ToggleTrayLED(){
  if(TrayLEDState == 1){
    TrayLEDState = 0;
    digitalWrite(TrayLEDPin, TrayLEDState);
  }
  else{
    TrayLEDState = 1;
    digitalWrite(TrayLEDPin, TrayLEDState);
  }
}

void getSerialConstant(){
 while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();

      switch (inChar) {
    case 'q':
       myKp += 1.0;
      break;
    case 'a':
       myKp -= 1.0;
       if(myKp<0){
        myKp = 0;
       }
      break;
    case 'w':
       myKi += 1.0;
      break;
    case 's':
       myKi -= 1.0;
       if(myKi<0){
        myKi = 0;
       }
```

```
      break;
    case 'e':
       myKd += 0.05;
      break;
    case 'd':
       myKd -= 0.05;
       if(myKd<0){
        myKd = 0;
       }
      break;
    case 'r':
       idealAngle += 0.01;
      break;
    case 'f':
       idealAngle -= 0.01;
      break;

    default: break;
    }
  }
  Serial.print(myKp); Serial.print("  ");
  Serial.print(myKi); Serial.print("  ");
  Serial.print(myKd); Serial.print("  ");
  Serial.println(Output); Serial.print("  ");
//  Serial.print(loopTime); Serial.print("  ");
//  Serial.print(currentAngle); Serial.print("  ");
//  Serial.print(accelAngle); Serial.print("  ");
//  Serial.print(Input); Serial.print("  ");
//  Serial.println(idealAngle);

//  Setpoint = idealAngle;
  BalancePID.SetTunings(myKp, myKi, myKd);
}
```

**Remote Control Code:**

This code is based on : YourDuinoStarter Example: nRF24L01

```
/*YourDuinoStarter Example: nRF24L01 Transmit Joystick values
 - WHAT IT DOES: Reads Analog values on A0, A1 and transmits
   them over a nRF24L01 Radio Link to another transceiver.
 - SEE the comments after "//" on each line below
 - CONNECTIONS: nRF24L01 Modules See:
 http://arduino-info.wikispaces.com/Nrf24L01-2.4GHz-HowTo
   1 - GND
   2 - VCC 3.3V !!! NOT 5V
   3 - CE to Arduino pin 8
   4 - CSN to Arduino pin 9
   5 - SCK to Arduino pin 13
   6 - MOSI to Arduino pin 11
   7 - MISO to Arduino pin 12
   8 - UNUSED
   -
   Analog Joystick or two 10K potentiometers:
   GND to Arduino GND
   VCC to Arduino +5V
   X Pot to Arduino A0
   Y Pot to Arduino A1

 - V1.00 11/26/13
   Based on examples at http://www.bajdi.com/
   Questions: terry@yourduino.com */

/*-----( Import needed libraries )-----*/
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <stdint.h>
/*-----( Declare Constants and Pin Numbers )-----*/
#define CE_PIN   8
#define CSN_PIN 9
#define JOYSTICK_X A0
#define JOYSTICK_Y A1
#define JOYSTICK_BUTTON 7
#define BUTTON1 6
#define BUTTON2 5
#define PWR_LED 3

// NOTE: the "LL" at the end of the constant is "LongLong" type
```

```
const uint64_t pipe = 0xE8E8F0F0E1LL; // Define the transmit pipe

int lasttime;    //used for timing of LED fading
signed int LEDVal = 0;    //used for keeping track of LED PWM duty cycle
bool countDir = 1;  //used for keeping track of whether counting up or down
bool lastB2state = 0;

/*-----( Declare objects )-----*/
RF24 radio(CE_PIN, CSN_PIN); // Create a Radio
/*-----( Declare Variables )-----*/
uint8_t inputArray[32];  // 5 element array holding Joystick and button readings

void setup()   /****** SETUP: RUNS ONCE ******/
{
// Serial.begin(9600);
  radio.begin();
  radio.openWritingPipe(pipe);

  pinMode(JOYSTICK_BUTTON, INPUT);
  pinMode(BUTTON1, INPUT);
  pinMode(BUTTON2, INPUT);
  pinMode(PWR_LED, OUTPUT);

  digitalWrite(PWR_LED, LOW);
  lasttime = millis();
  initializeArrayZeros(inputArray);
}//--(end setup )---


void loop()   /****** LOOP: RUNS CONSTANTLY ******/
{
  inputArray[0] = analogRead(JOYSTICK_X)/4;
  inputArray[1] = 255-analogRead(JOYSTICK_Y)/4;    //255-y to invert
  inputArray[2] = digitalRead(BUTTON1);

    if(lastB2state == 1){
      inputArray[3] = 0;
    }
    else{
    lastB2state = digitalRead(BUTTON2);
    inputArray[3] = lastB2state;
    }

  inputArray[4] = digitalRead(JOYSTICK_BUTTON);
```

```
  if(inputArray[0] == 0){    //don't send a 0 in this byte. Used to determine whether remote is
sending data.
   inputArray[0] = 1;
  }

  radio.write( inputArray, sizeof(inputArray) );

//  Serial.print(inputArray[0]); Serial.print("  ");
//  Serial.print(inputArray[1]); Serial.print("  ");
//  Serial.print(inputArray[2]); Serial.print("  ");
//  Serial.print(inputArray[3]); Serial.print("  ");
//  Serial.print(inputArray[4]); Serial.print("  ");
//  Serial.print(inputArray[5]); Serial.print("  ");
//  Serial.print(inputArray[6]); Serial.print("  ");
//  Serial.print(inputArray[7]); Serial.print("  ");
//  Serial.println(inputArray[8]);

  UpdateStatusLED();


}//--(end main loop )---

/*-----( Declare User-written Functions )-----*/
void initializeArrayZeros(uint8_t* array){
  for(int i=0; i<sizeof(array); i++){
   array[i] = 0;
  }
  return;
}

void UpdateStatusLED(){
  if(millis()-lasttime > 10){
     lasttime = millis();
    if(countDir == 1){      //counting up?
      if(LEDVal < 255){
        LEDVal += 4;
      }
     }

    if(countDir == 0){      //counting down?
     if(LEDVal > 0){
       LEDVal -= 4;
     }
    }

     if(LEDVal > 255){    //check high limit
```

```
      LEDVal = 255;
      countDir = 0;
     }
    if(LEDVal < 0){     //check low limit
      LEDVal = 0;
      countDir = 1;
      }
   analogWrite(PWR_LED, LEDVal);
  }
  return;
}    //END UpdateStatusLED()

//NONE
//*********( THE END )***********
```

**LED Tray Code:**

```
'*****************************************************************
'*  Name    : TrayLED.PBP                                        *
'*  Author  : [select VIEW...EDITOR OPTIONS]                     *
'*  Notice  : Copyright (c) 2014 [select VIEW...EDITOR OPTIONS]  *
'*          : All Rights Reserved                                *
'*  Date    : 3/3/2014                                           *
'*  Version : 1.0                                                *
'*  Notes   :                                                    *
'*          :                                                    *
'*****************************************************************

' PIC16F88 code template for MECH307 Labs

' The following configuration bits and register settings
' enable the internal oscillator, set it to 8MHz,
' disables master clear, and turn off A/D conversion

' Configuration Bit Settings:
' Oscillator                        INTRC (INT102) (RA6 for I/O)
' Watchdog Timer                    Enabled
' Power-up Timer                    Enabled
' MCLR Pin Function                 Input Pin (RA5 for I/O)
' Brown-out Reset                   Enabled
' Low Voltage Programming           Disabled
' Flash Program Memory Write              Enabled
' CCP Multiplexed With                    RB0
' Code                              Not Protected
' Data EEPROM                       Not Protected
' Fail-safe Clock Monitor           Enabled
' Internal External Switch Over           Enabled

' Define configuration settings (different from defaults)
#CONFIG
   __CONFIG _CONFIG1, _INTRC_IO & _PWRTE_ON & _MCLR_OFF & _LVP_OFF
#ENDCONFIG

' Set the internal oscillator frequency to 8 MHz
DEFINE OSC 8
OSCCON.4 = 1
OSCCON.5 = 1
OSCCON.6 = 1

' Turn off the analog to digital converters. Refer to Threadd Design Example A.4
' in the textbook foran example of how to configure and use A/D conversion
ansel = 0
```

```
'Configure Port A and B as outputs
TrisA = %00100000
TrisB = %01000000

	gosub InitializeLEDs
' Put your code here:
myloop:
	If (PortB.6==0) then
	gosub LED_loop
	endif
	Goto myloop	'go back to label "loop" repeatedly


LED_loop:

	High PORTA.2	'turn on PORTA.2
	high PORTB.0	'turn on LED connected to PORTB.0
	pause 100	'delay for 100 milliseconds

	low PORTA.2	'turn off PORTA.2
	Low PORTB.0	'turn off LED connected to PORTB.0

	High PORTA.3
	high PORTB.1
	pause 100

	low PORTA.3
	Low PORTB.1

	High PORTA.4
	high PORTB.2
	pause 100

	low PORTA.4
	Low PORTB.2

	High PORTA.1
	high PORTB.3
	pause 100

	low PORTA.1
	Low PORTB.3

	High PORTA.0
	high PORTB.4
```

```
        pause 100

        low PORTA.0
        Low PORTB.4

        High PORTA.6
        high PORTB.5
        pause 100

        low PORTA.6
        Low PORTB.5

        Return

InitializeLEDs:

        low PORTA.2     'turn off PORTA.2
        Low PORTB.0     'turn off LED connected to PORTB.0
        low PORTA.3
        Low PORTB.1
        low PORTA.4
        Low PORTB.2
        low PORTA.1
        Low PORTB.3
        low PORTA.0
        Low PORTB.4
        low PORTA.6
        Low PORTB.5

        Return

        end
```

## References


- PID Library
  - Brett Beauregard
  - http://playground.arduino.cc/Code/PIDLibrary

- Wire Library
  - Standard Arduino Library

- I2C library (non-standard wire library for Arduino)
  - Jeff Rowberg
  - http://www.i2cdevlib.com/

- MPU-6050 library
  - Jeff Rowberg
  - http://www.i2cdevlib.com/devices/mpu6050#source

- Math library (arctan2() function)
  - http://arduino.cc/en/Math/H

- LCD library
  - Standard Arduino Library

- SPI Library
  - Standard Arduino Library

- Wireless Library
  - J. Coliz
  - http://arduino-info.wikispaces.com/nRF24L01-RF24-Examples

- Standard data size library
  - Standard Arduino Library