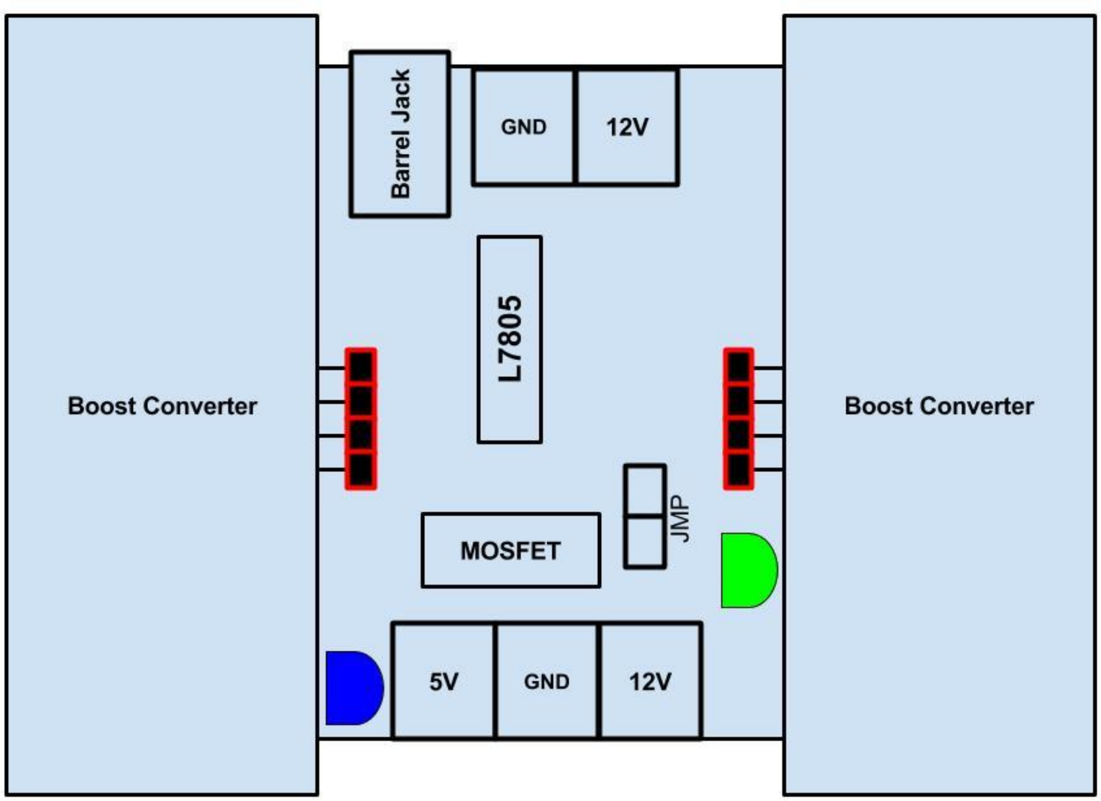


Barrel Jack
GND
12V
L7805
Boost Converter
Boost Converter
JMP
MOSFET
5V
GND
12V

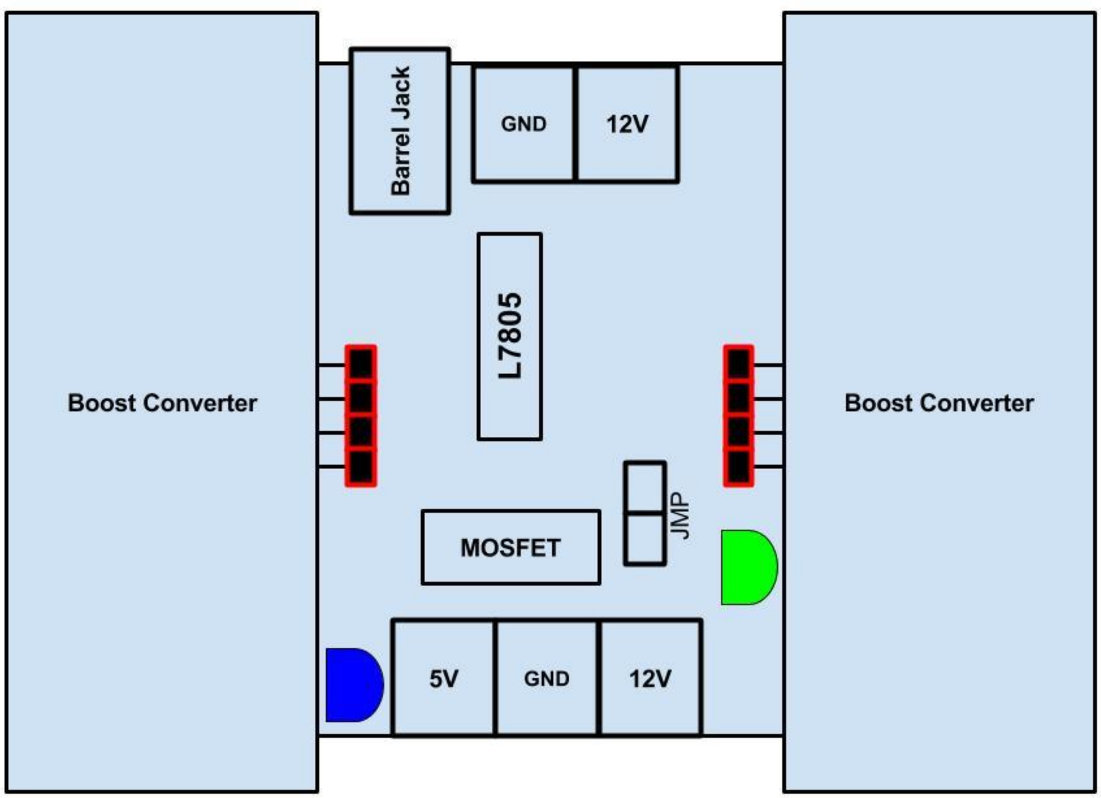
Autonomous Nerf Tank
(A.N.T.)

EE400D Senior Design
Instructor: Dr. Aftab Ahmed
Date: 08/17/2017

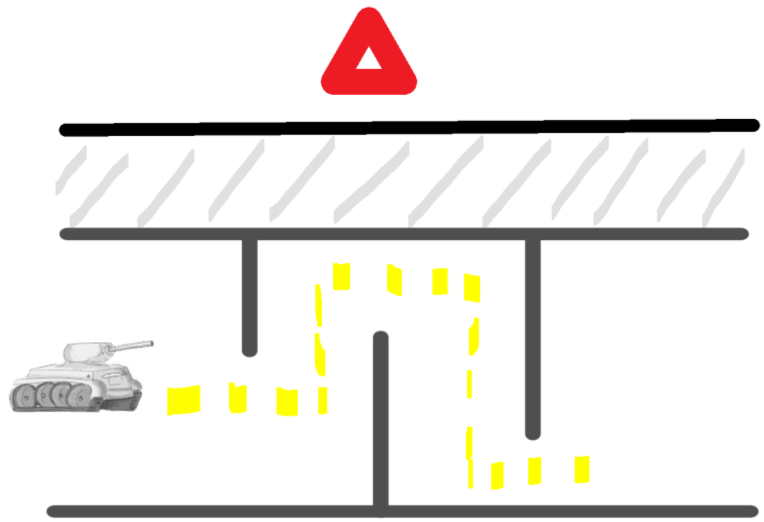
Team #3
Ivan Lopez
Erik Cuevas
Sadaf Sadat Riazi
Zackery Kuntz

ABSTRACT

This senior design project applies DSP and embedded systems knowledge to create an Autonomous Nerf Tank (A.N.T). With the implementation of off-the-shelf electronics and



MATLAB base image processing, the A.N.T. completes three tasks: 1) Navigate a maze . 2) Identify the target. 3) Request permission to fire from user and strike the target if user allows.



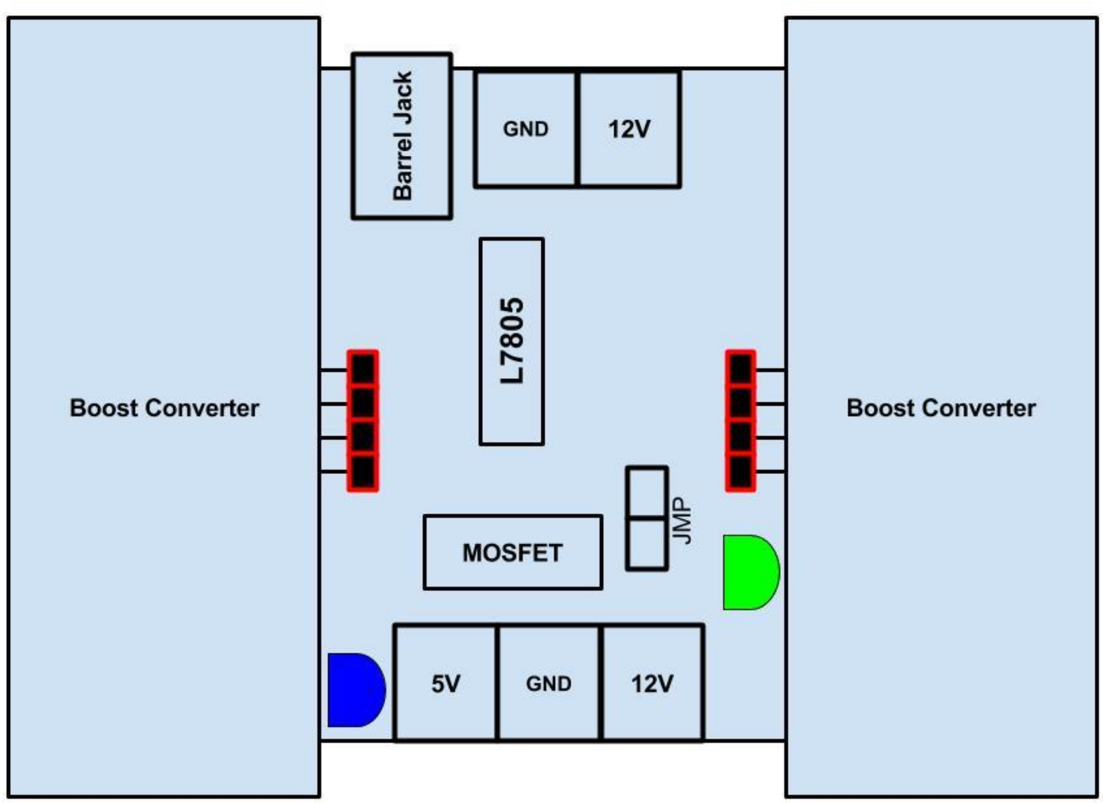
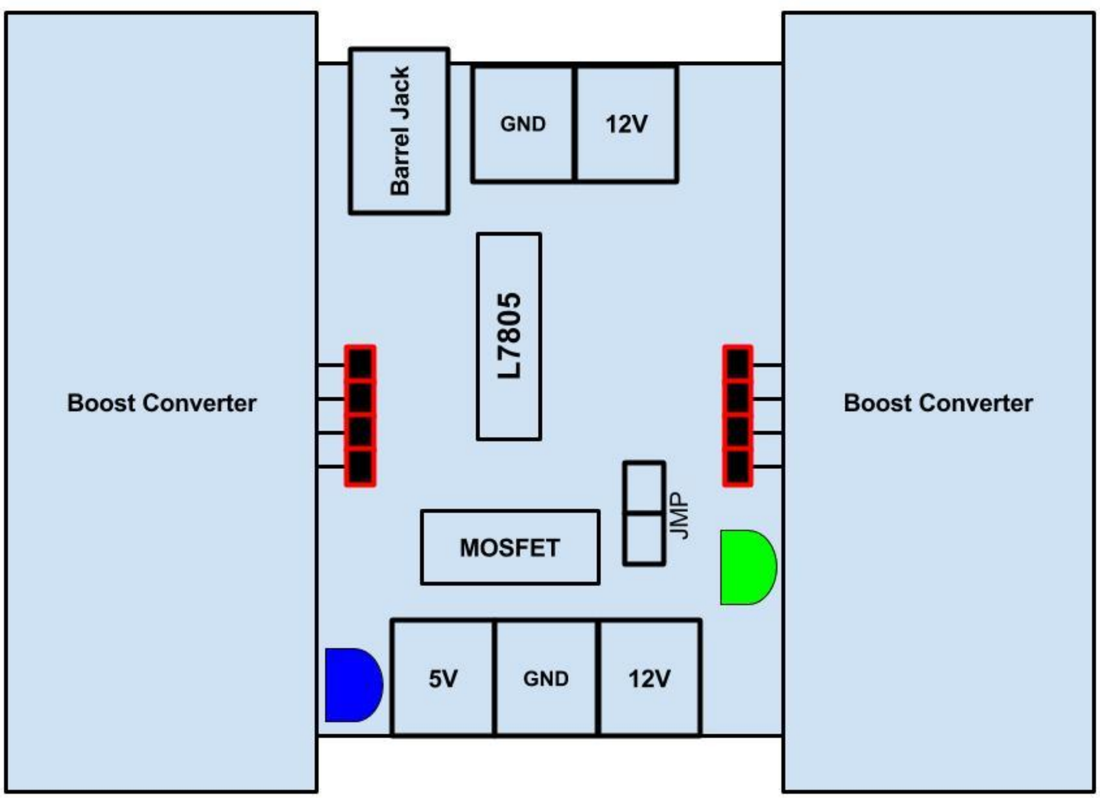


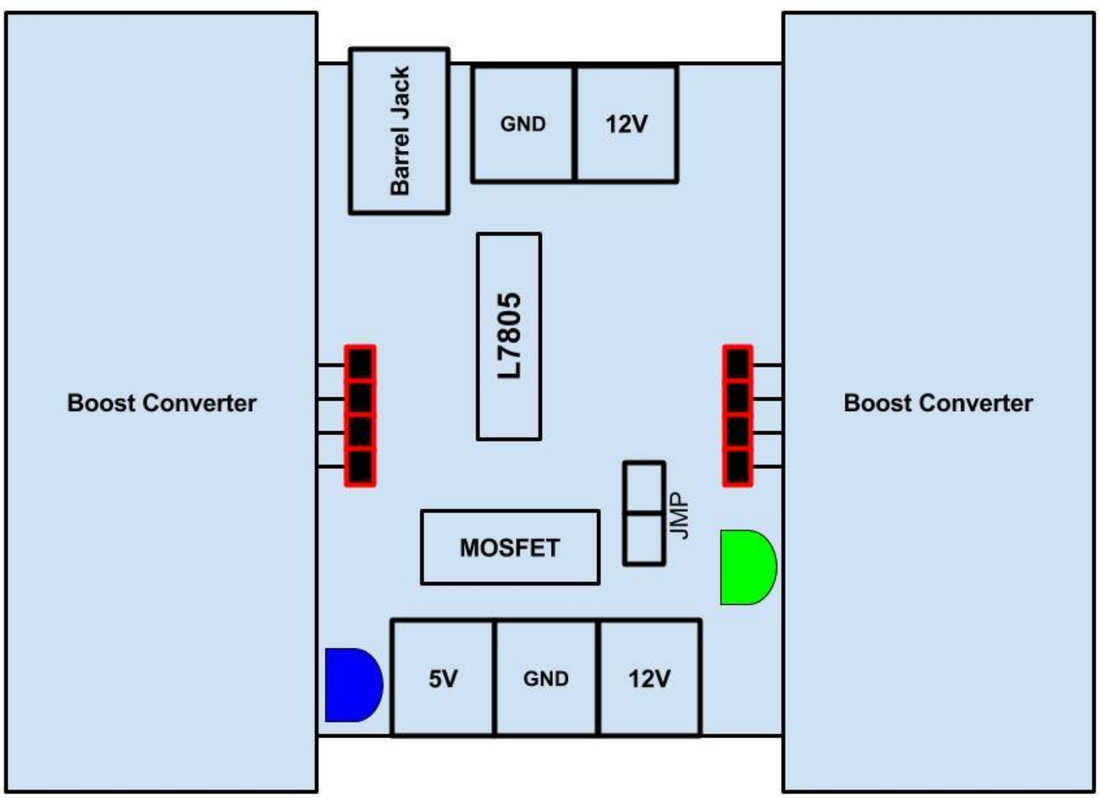
Fig. 1 A.N.T. Concept Art

Contents



3.5 Autonomous movement

3.2 Physical layout



1. Introduction

The ANT design is part of the senior design course 400D. It aims to help electrical engineers become familiar with the engineering method and how to implement it to produce measurable results. The tank is designed to autonomously navigate a walled maze and locate a previously determined target and fire a dart to the target.

2. Design Requirements

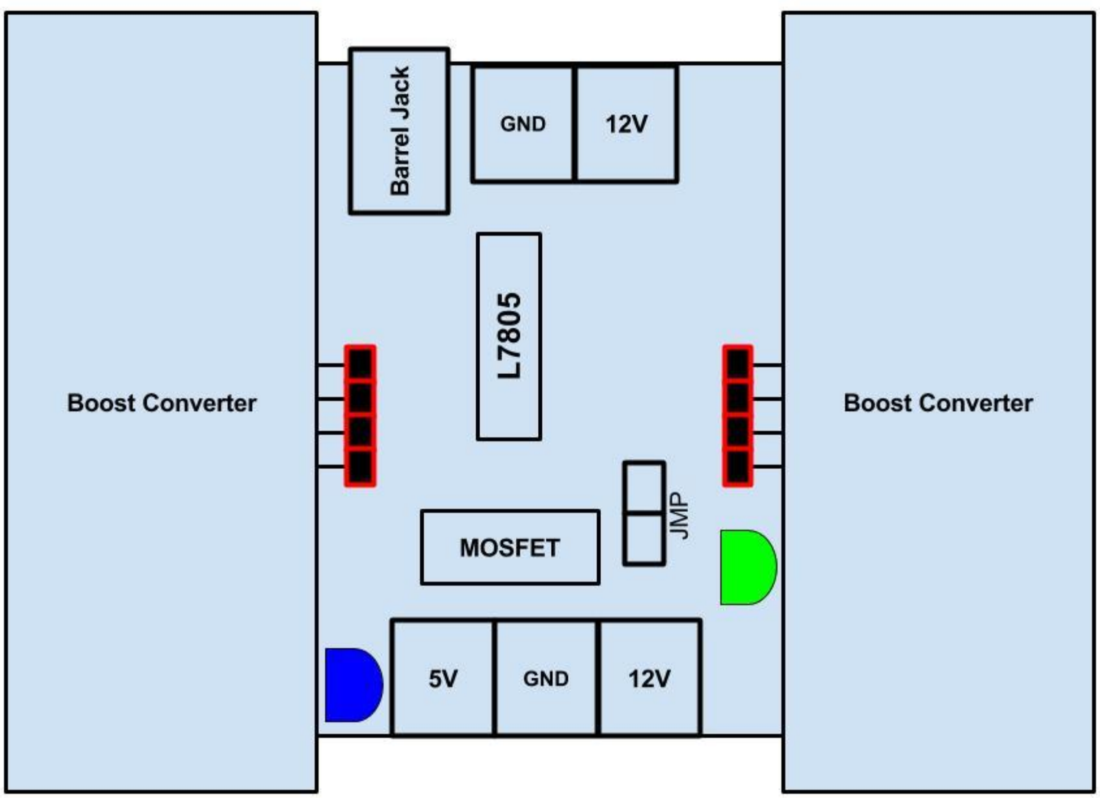
The system requirements are as follow:

<u>Tank Movement</u>
The vehicle is able to move forward and backward, turn both left and right, and stop.

<u>Automation</u>
The tank is able to navigate the maze without human assistance.

<u>Target Acquisition and tracking</u>
The tank is able to locate and amin onto a predetermined static target.

<u>Firing</u>



The tank is able, once the target is within range, to ask for permission to fire and once permission is granted the tank is able to fire the projectile to the target.

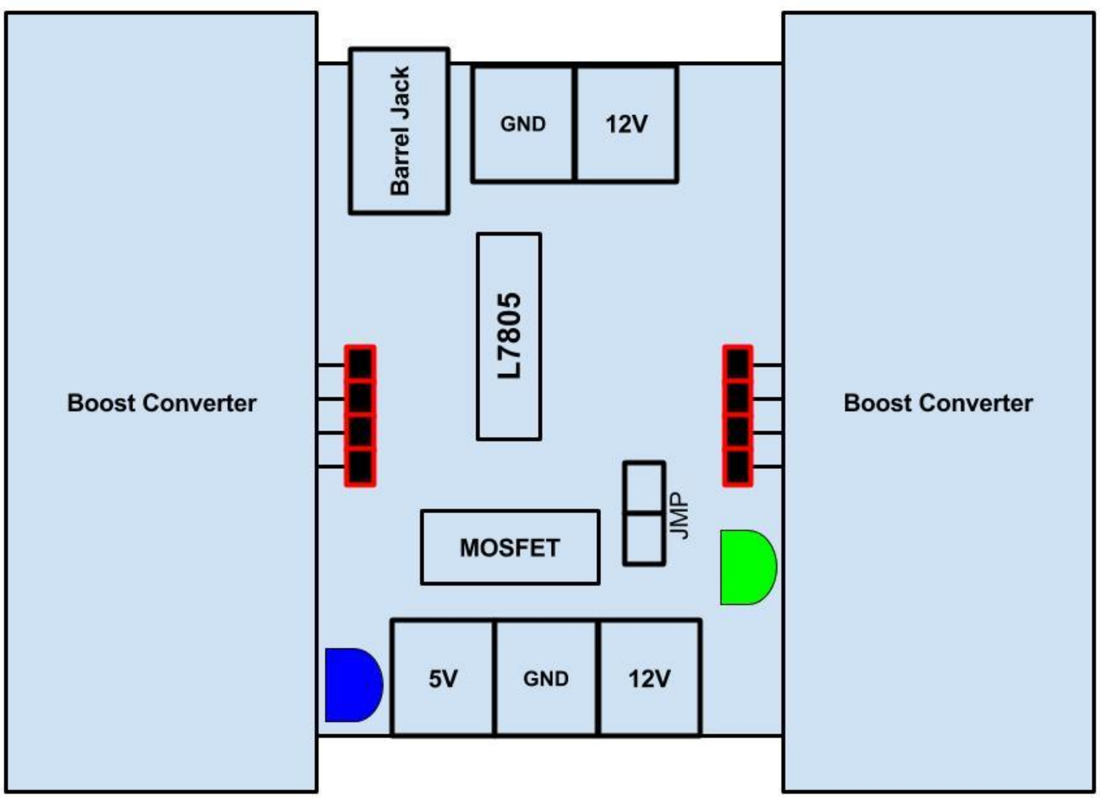
3. Design and Implementation:

In order to realize the design a Arduino 2560 Mega is used to provide the logic control for the movement of the rover and the the aiming and firing system. This particular board is equipped with a Atmega2560 chip which can provide up to 16 MIPS at 16MHz.

The Arduino board is a well reputed developmental board which comes with its own IDE, Integrated Development Environment. It uses C programing language and provides its own compiler.

The Arduino 2560 Mega will provide the input and outputs ports to implement the movement of the tank and the firing of the projectile.

The movement will be realized with the use of two 12 VDC motors connected to a H bridge

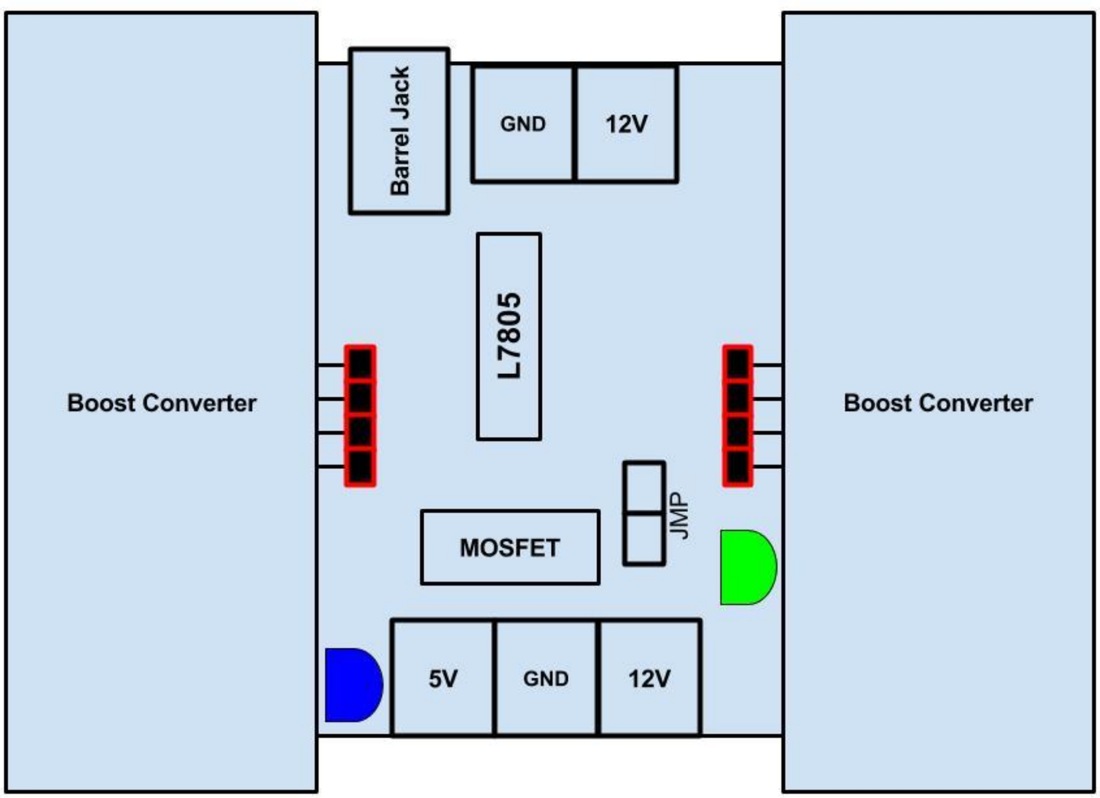


circuit which will provide the necessary current for the motors to drive the tank. The H bridge is used because the Arduino board does not provide the enough current to drive the motors.

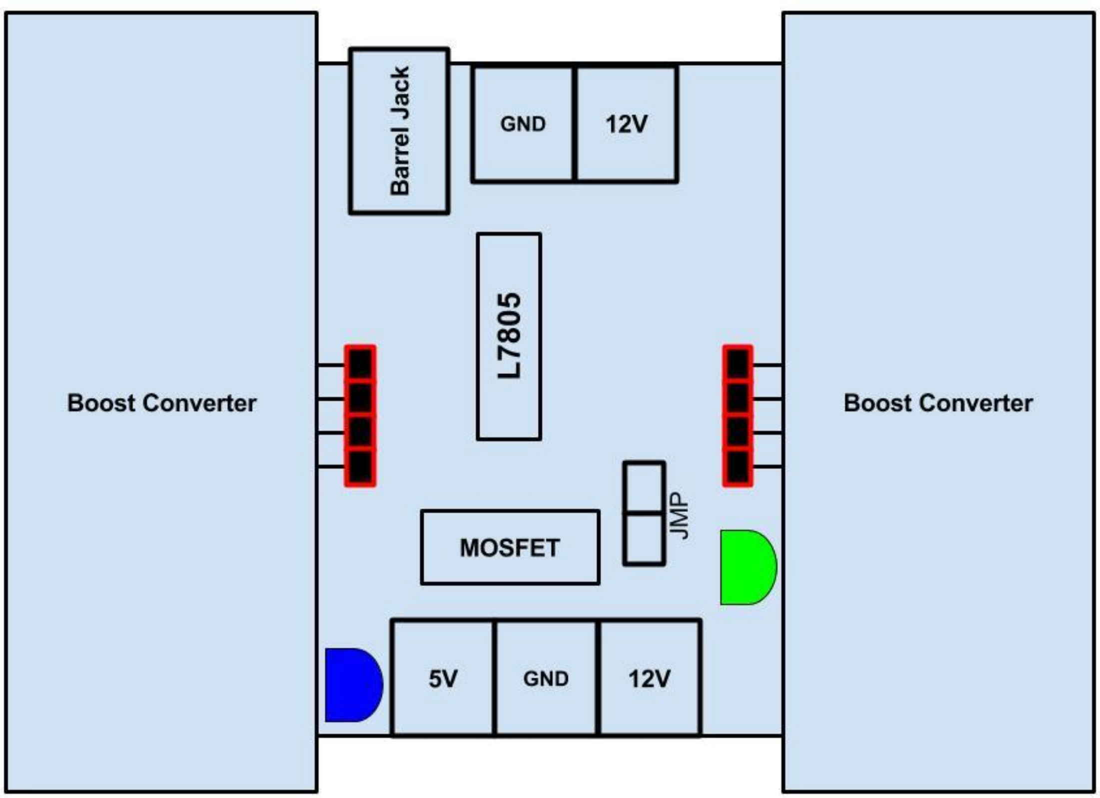
The tank will be guided by three supersonic sensors that will provide the system with the proximity of obstacles to the tank. The Arduino will take these inputs and decide which way to navigate.

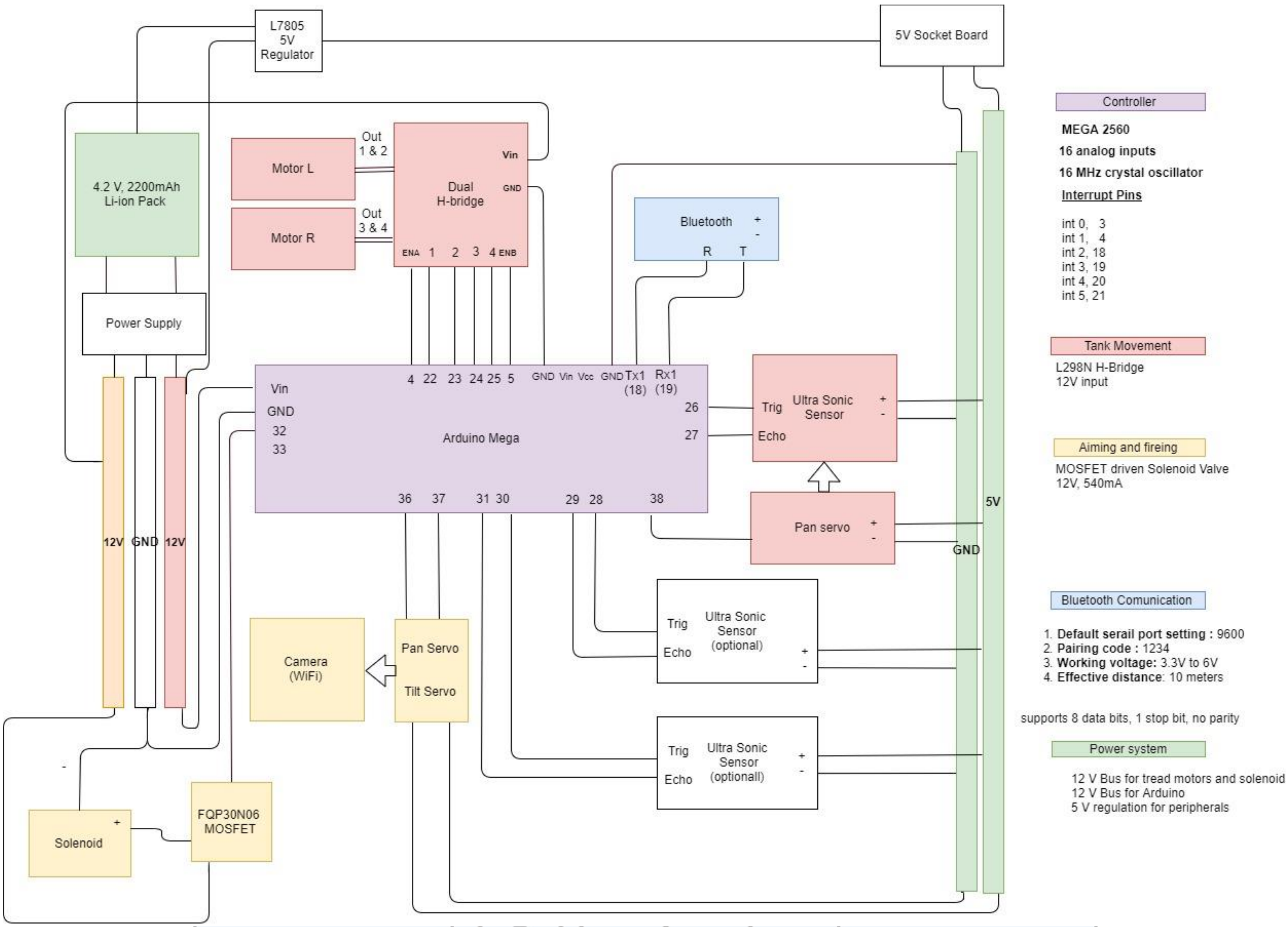
The image processing will take place on a remote station with the assistance of MATLAB. The microcontroller board does not have enough processing power to handle the image processing. In order to have the station do the image processing, the tank will transmit the image via wifi with the assistance of an ArduCam ESP8266 UNO and a ArduCam Mini 2MP module. The module is used to acquire the image on jpeg format and the ESP8266 board is used to create a access point with server capabilities to stream video.

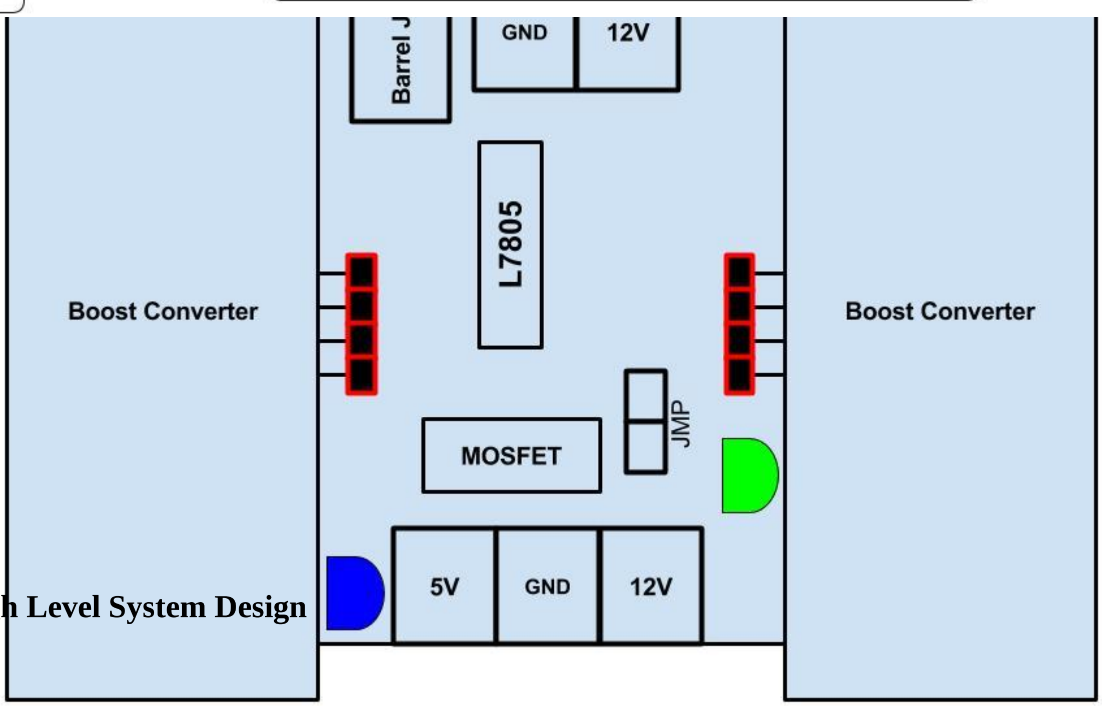
The firing command and target lock function will be implemented with MATLAB logic sent to the arduino board via a HC-O6 Bluetooth chip and 3 servo motors driven directly from the



Arduino board.





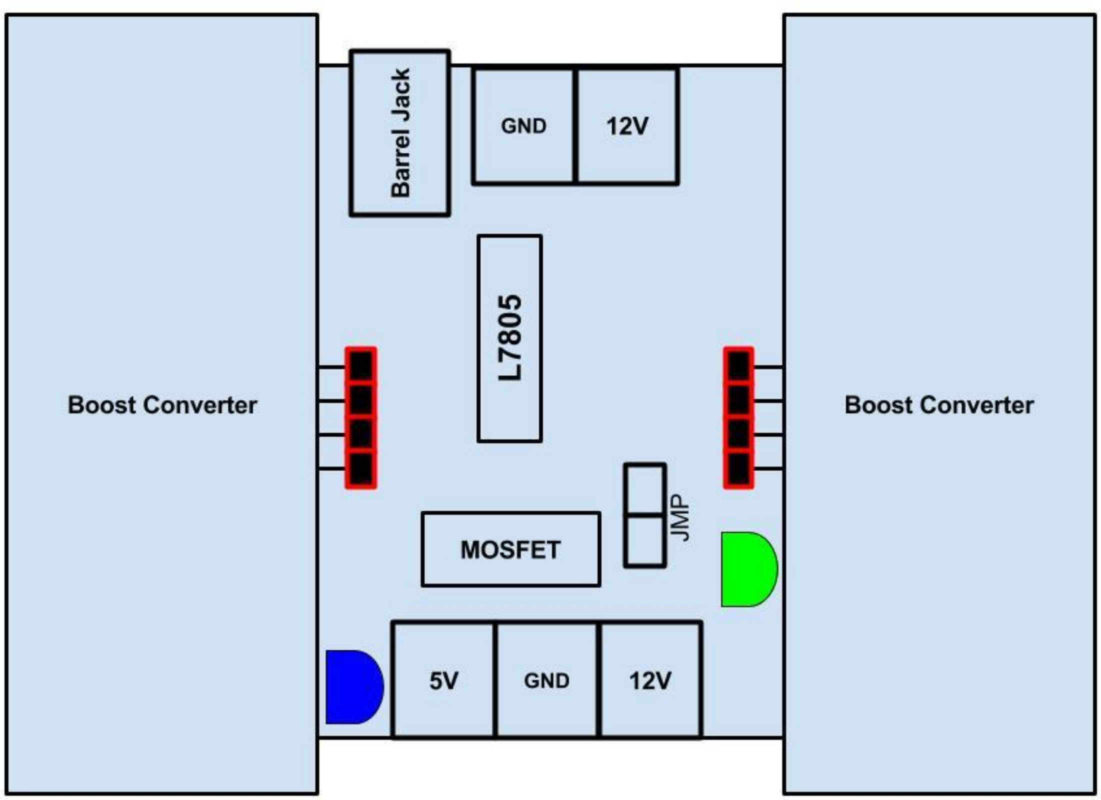


High Level System Design

3.3 Computer vision for Aiming Automation

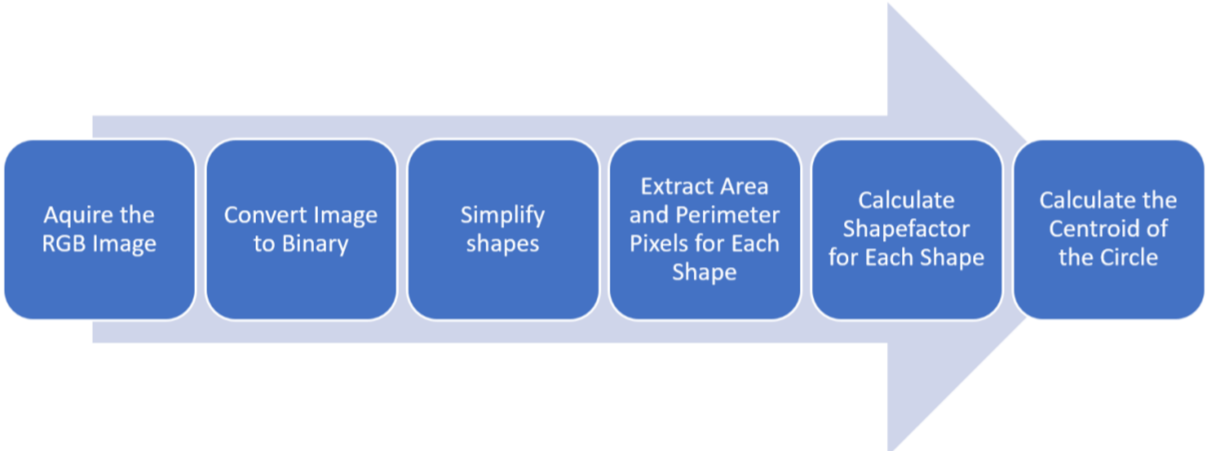
Scope

The A.N.T. has the ability to recognize a sub-set of shapes and colors which can be set as targets in the matlab interface. Recognizable shapes are circles and Color option available is Red. In addition to shape and color detection, the A.N.T. can leverage MATLAB computer vision toolbox for face recognition and use it as an alternative target detection. This option is also available via the matlab user interface.



Shape Detection Theory

The figure below breaks down the shape detection digital image processing into six steps

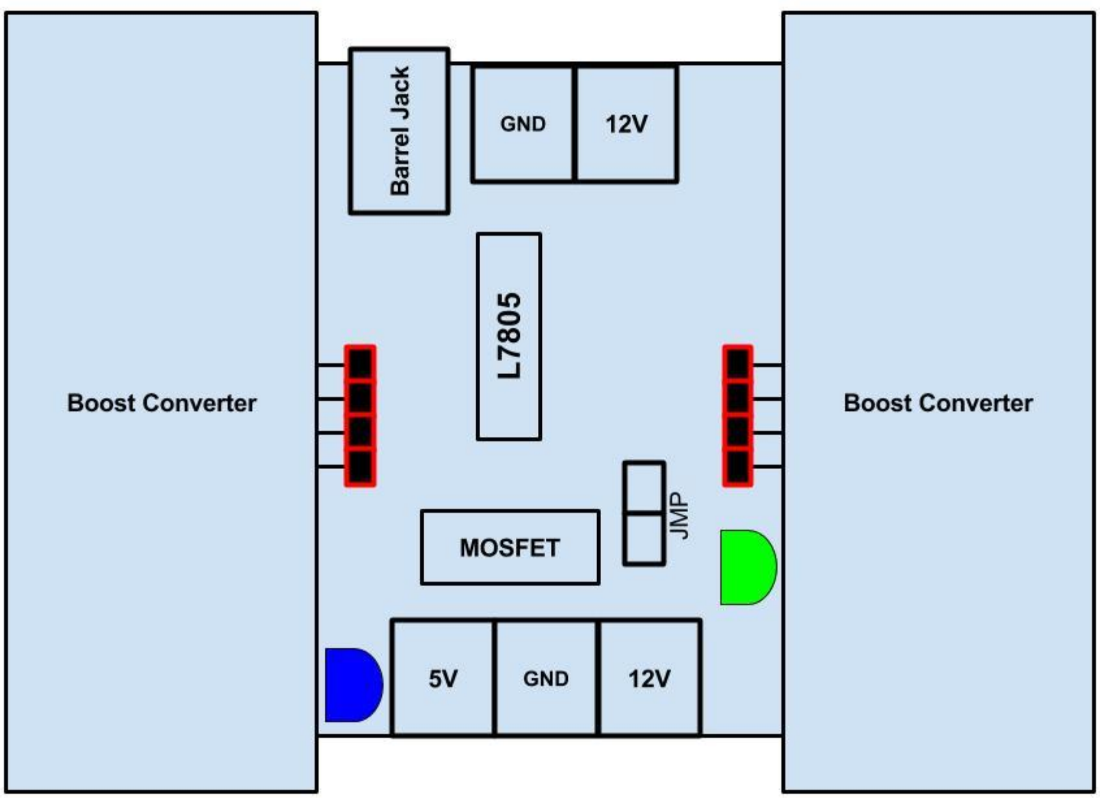


Going from RGB to Binary

The first step in going from RGB to binary is to convert the image to grayscale.

Matlab can do this using the rgb2gray function. The rgb2gray function takes in the 3 matrices for the red, green and blue channels and flattens them into a single matrix by taking the average of them. Then to convert the grayscale image to a binary image a threshold is shosen; Any pixel below that threshold will be set to black and any pixel above the threshold is set to white.

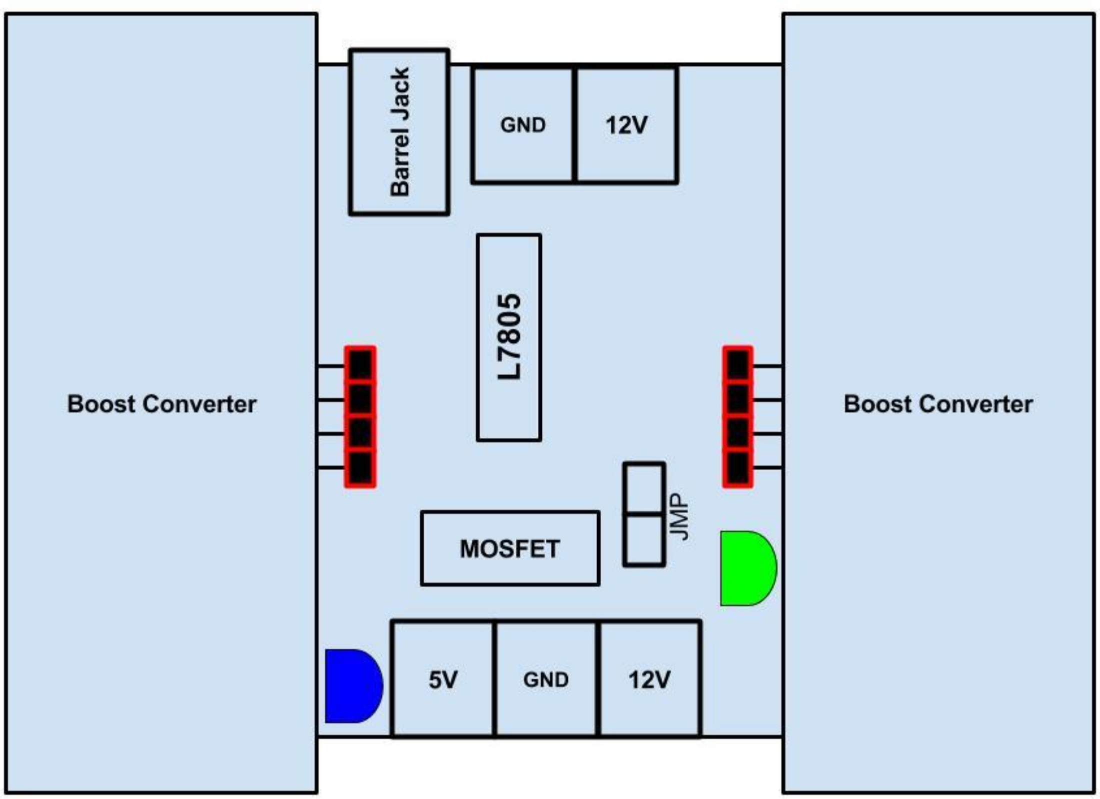
To choose the threshold level we can apply the Otsu's Algorithm with the `graythresh` function.



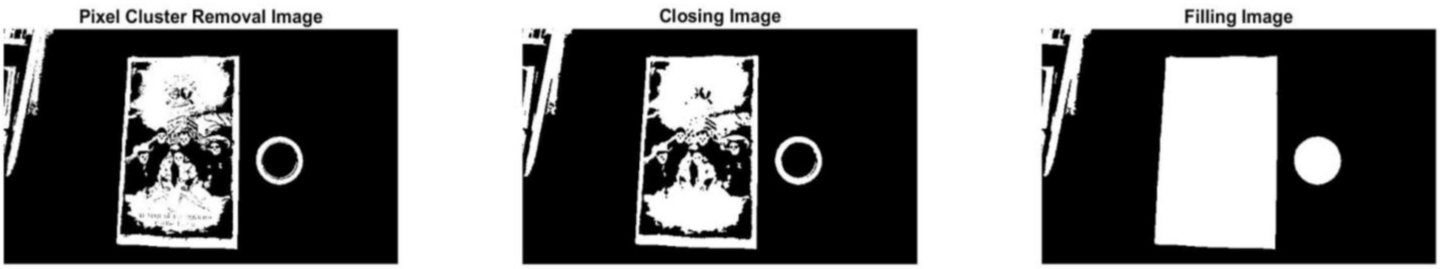
MATLAB RGB to Binary Code

```
%% Read RGB Image & convert to Binary -----------------------------------------------
cam=ipcam('http://192.168.4.1/video.mjpeg'); % Create an IP Camera object
rgb_Image=snapshot(cam);             %  take a snapshot of the video feed
[y,x,z]=size(I);                      %  save image dimension
gray_Image=rgb2gray(rgb_Image);      %  convert to grayscale
threshold= graythresh(gray_Image);   %  computes a global threshold(Otsu's method)
bw=imbinarize(I,threshold);          %  convert an intensity image to a binary image
```

Cleaning and simplifying the Image



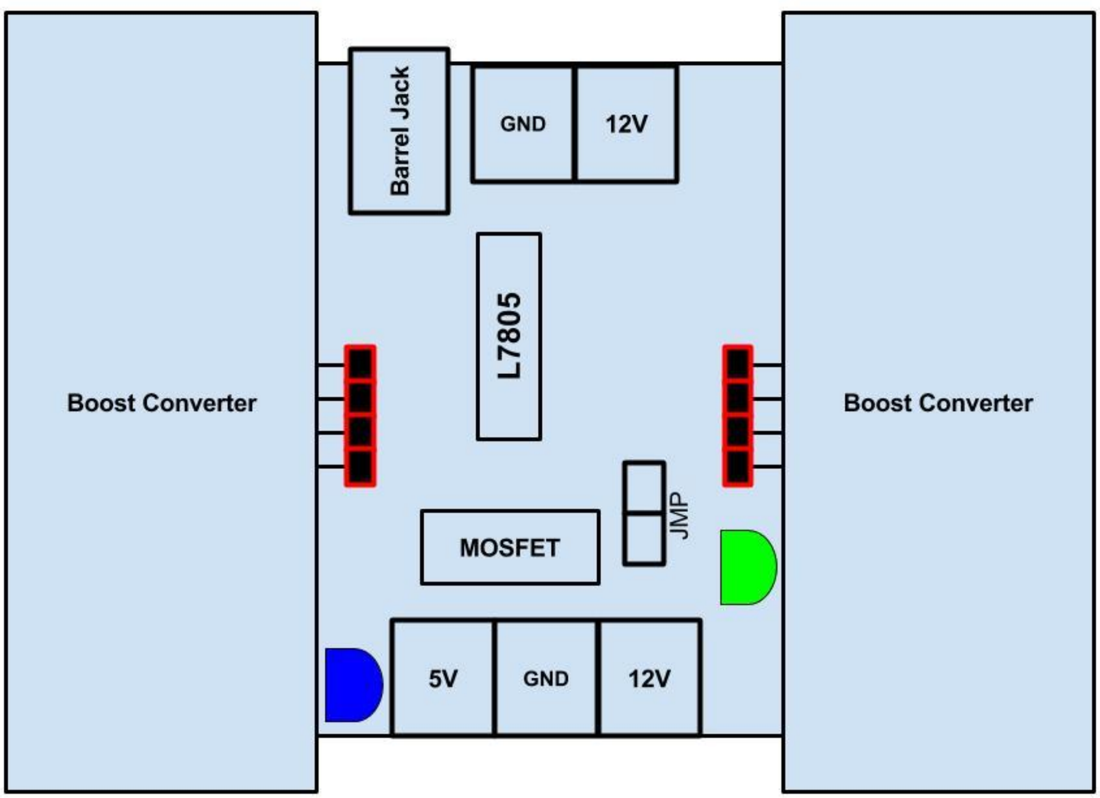
The next part of the process is to simplify the image by removing small clusters of pixels and



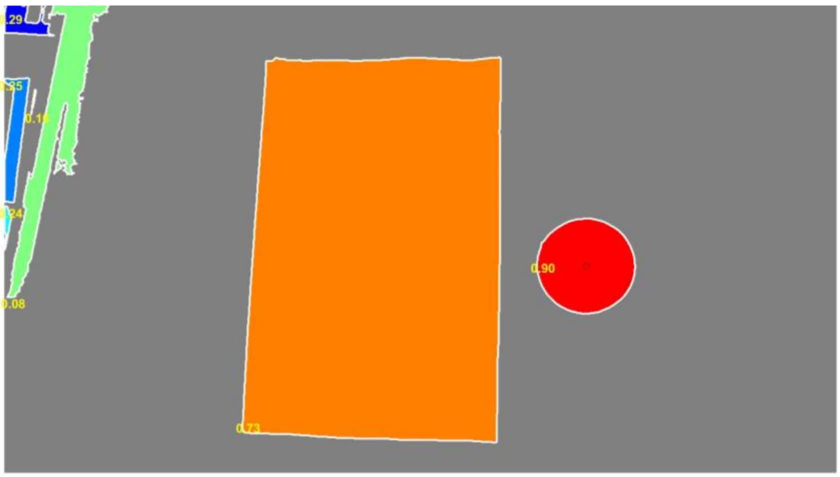
applying dilation and erosion to fill shapes into solid shapes.

<u>MATLAB Cleaning and simplifying the Image Code</u>

```
%% Remove the Noise with morphology functions ---------------------------------------
bw=bwareaopen(bw,30);      % remover objects containing fewer than 30 pixels
SE = strel('disk',2);       % creates a disk-shaped structuring element,radius of 2.
bw=imclose(bw,SE);         % performs dialationa and errotions
bw=imfill(bw,'holes');     % fill holes
```

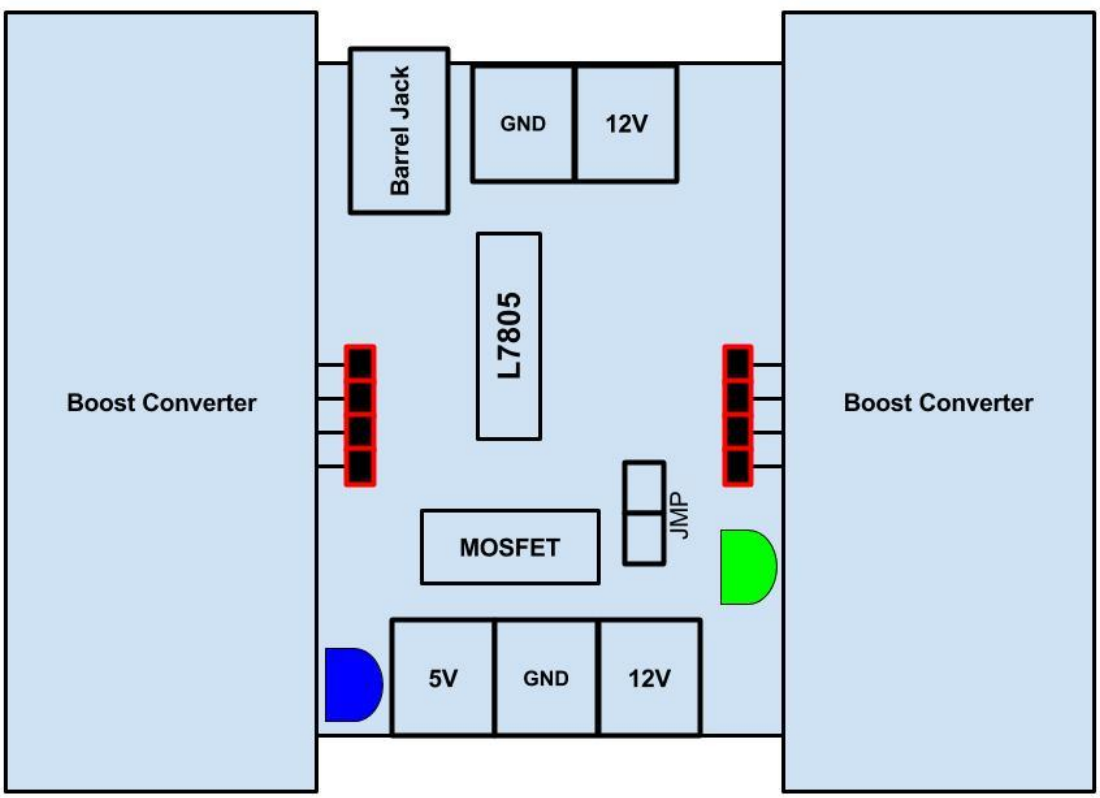


The next step is to separate each cluster of pixels as their own shape and determine their area and perimeter. Then ,use the a area and perimeter calculate the shape factor and determine which shapes are a circle. Figure ------on page (8) is the final image MATLAB sees with each different cluster of pixel represented by a different color and its shape factor displayed in yellow.



Calculating an Object's Circularity Shape factor

A circle has two unique equations for its area and perimeter.



$$A = \pi r^2 \quad \& \quad P = 2\pi r$$

These equations can be manipulated algebraically to relate its perimeter to its area

$$A = \frac{P^2}{4\pi}$$

Then we can then normalize this expression to have it equal one,

$$\frac{4\pi A}{P^2} = 1$$

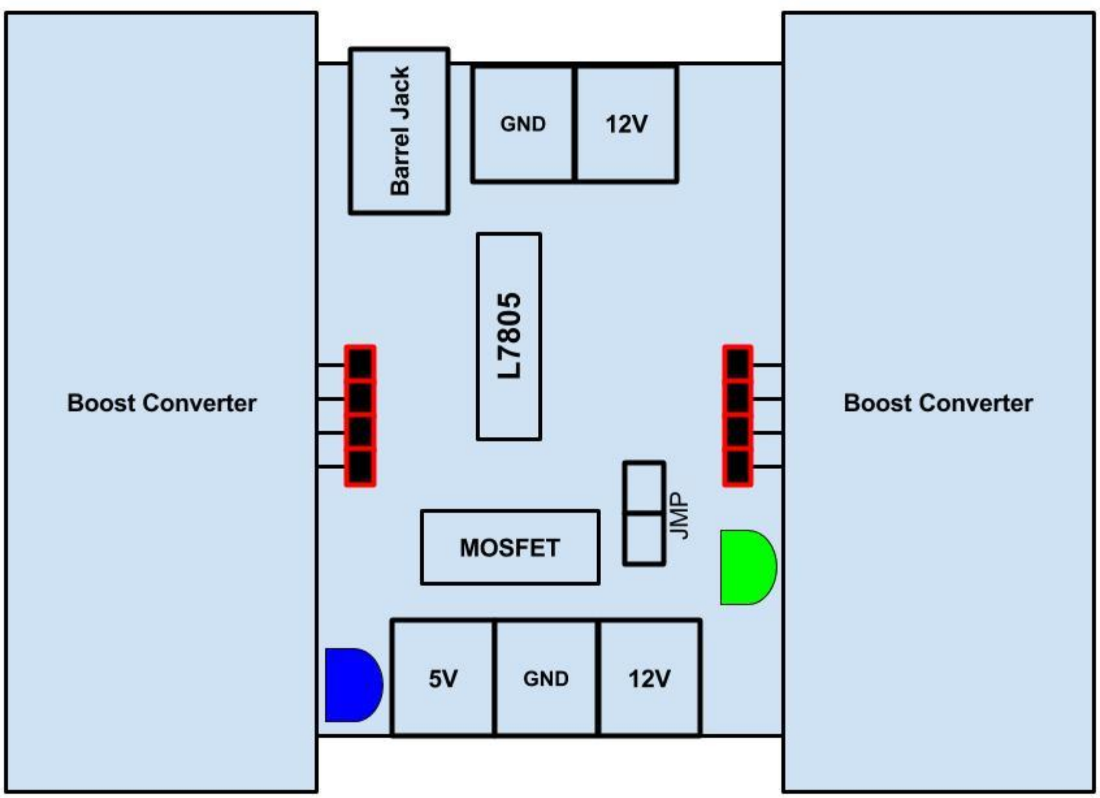
Therefore, if we have a shape's area and perimeter we can determine its circularity

$$\boxed{\text{Circularity Shape factor} = \frac{(P)^2}{2\pi A}}$$

The closer this number is to one the more circular the object is.

<u>Matlab Code</u>

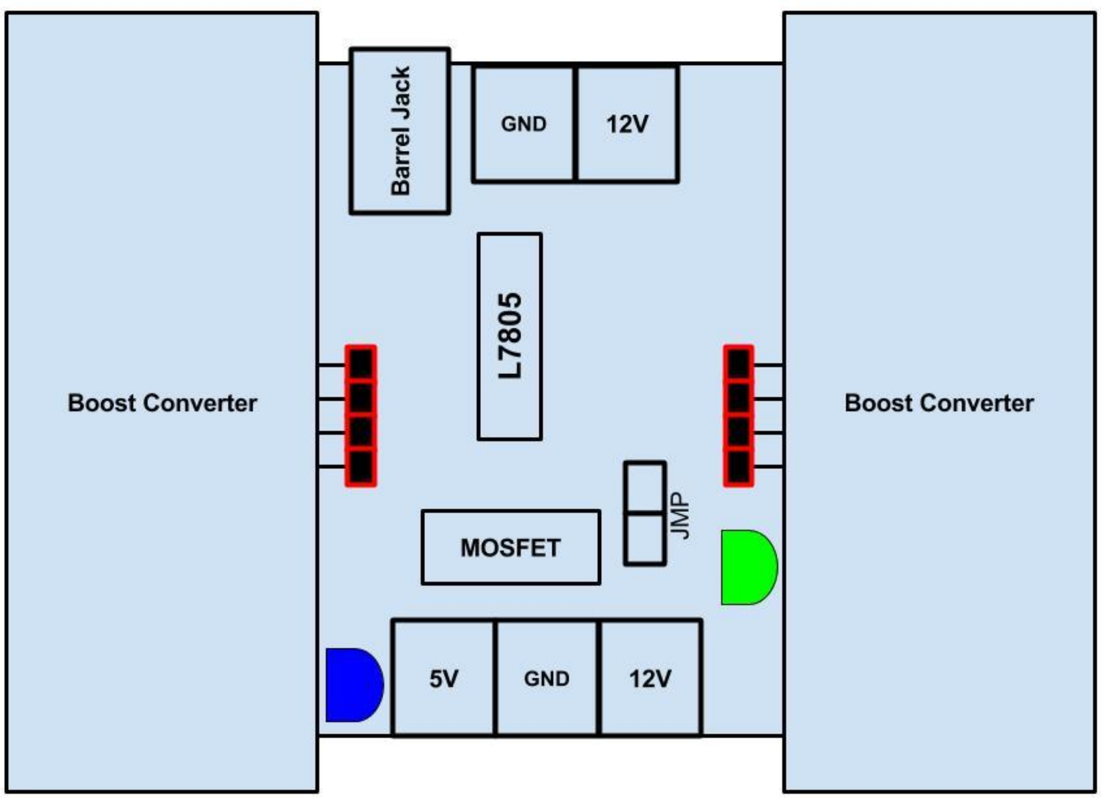
```
%% Create boundary matrix-------------------------------------------------------------
[edgeP,areaP]=bwboundaries(bw,'nohole');    % eged & area pixels matrixes of all shapes
%% Extract information from the boundaries--------------------------------------------
stats=regionprops(areaP,'Area','Centroid');% Get area and centroids for all shapes
```



```
threshold=0.87;                              % chosen after experimental trials
for k=1:length(edgeP)
        boundary=edgeP{k}; % get X and Y values for each boundary
        % Calculate its perimeter and area
        delta_sq=diff(boundary).^2;
        perimeter=sum(sqrt(sum(delta_sq,2)));
        area=stats(k).Area;
        % compute roundness metric
        metric=4*pi*area/perimeter^2;
        % save the centroid if its a circle
        if metric>threshold
                centroid=stats(k).Centroid;
        end
End
```

Color Recognition:

Color recognition is implemented on MATLAB with the use of the image processing toolbox. The HSV function is used to extract the hue, saturation and value of the individual pixels inside a rgb captured image. The a threshold function is created to preserve only the wanted values. These values vary from color to color and can be found readily through a web search. This threshold is applied to the three matrices composing the image. Then this three images are then logically anded to create a color mask. The color mask is further filtered by removing small clusters of pixels and filling the image with the color most likely to represent that color. This mask is then converted back to rgb and anded with the original image. This will cause all the non desire colors to disappear.



Aiming Results

Using the centroid of the target Matlab then sends a command to the panning stepper motor and the tilting servo motor to move the camera and nerf barrel unitil the centroid of the target matches the center of the frame. Below you can see the results before and after aming.

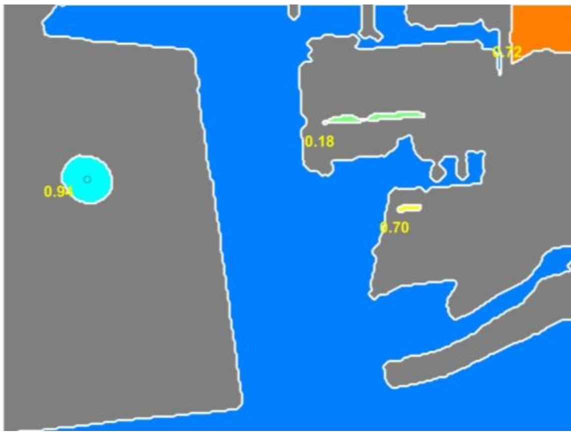


Fig 1 Off Center target light blue

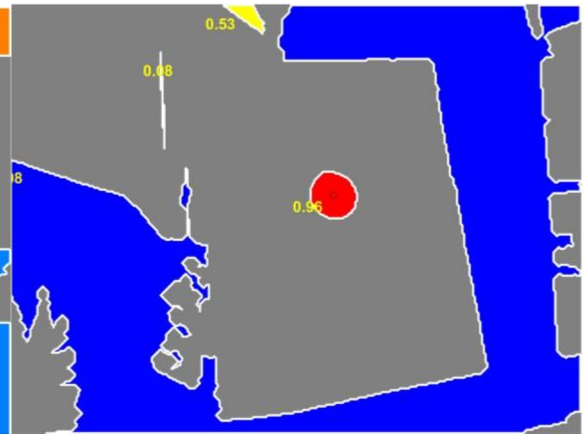
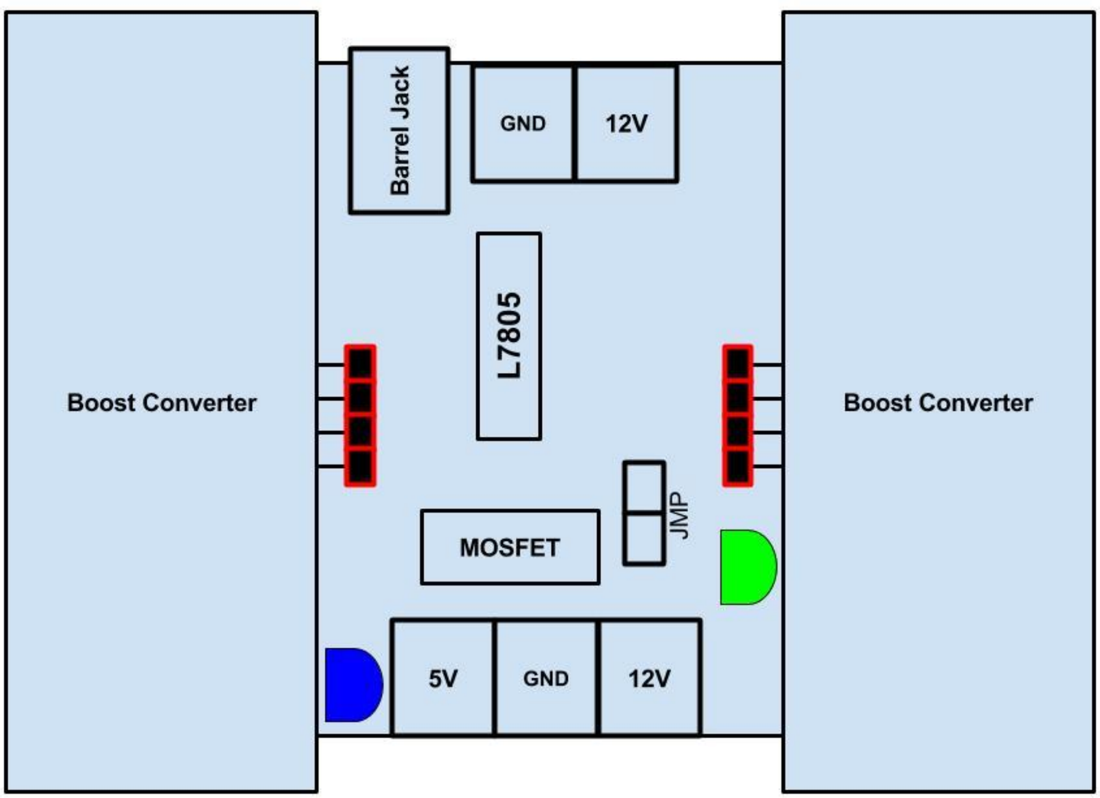


Fig 2 Amining function applied

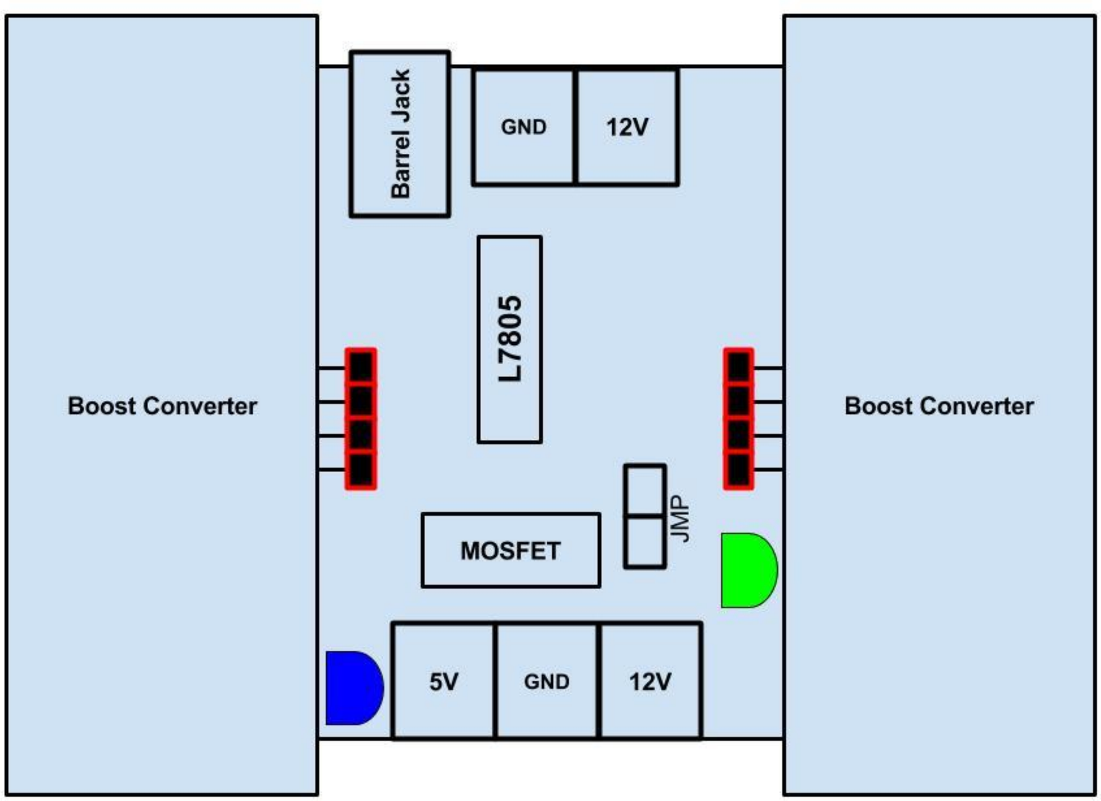
3.6 Wireless Communications

The wireless communication will be provided by a ESP8266, embedded on a AduCam ESP6266 UNO board, is a wifi integrated chip capable of both transmitting and receiving data to the rate



of 20Mbits/second with frequencies of 2.4G to 2.5G. The chip is set to run as an access point and a stream server. The server must be set as per the ipcam requirements in MATLAB. It needs to be able to communicate via HTTP (Hypertext Transfer Protocol) which is the industry standard for communication on the world wide web. The IP address has been been harcode onto the chip in order to be able to connect to the camera server without the need to find the address everytime the code is run (192.168.4.1) and the web server needs to be name in accordance with the ipcam (/video.mjpeg).

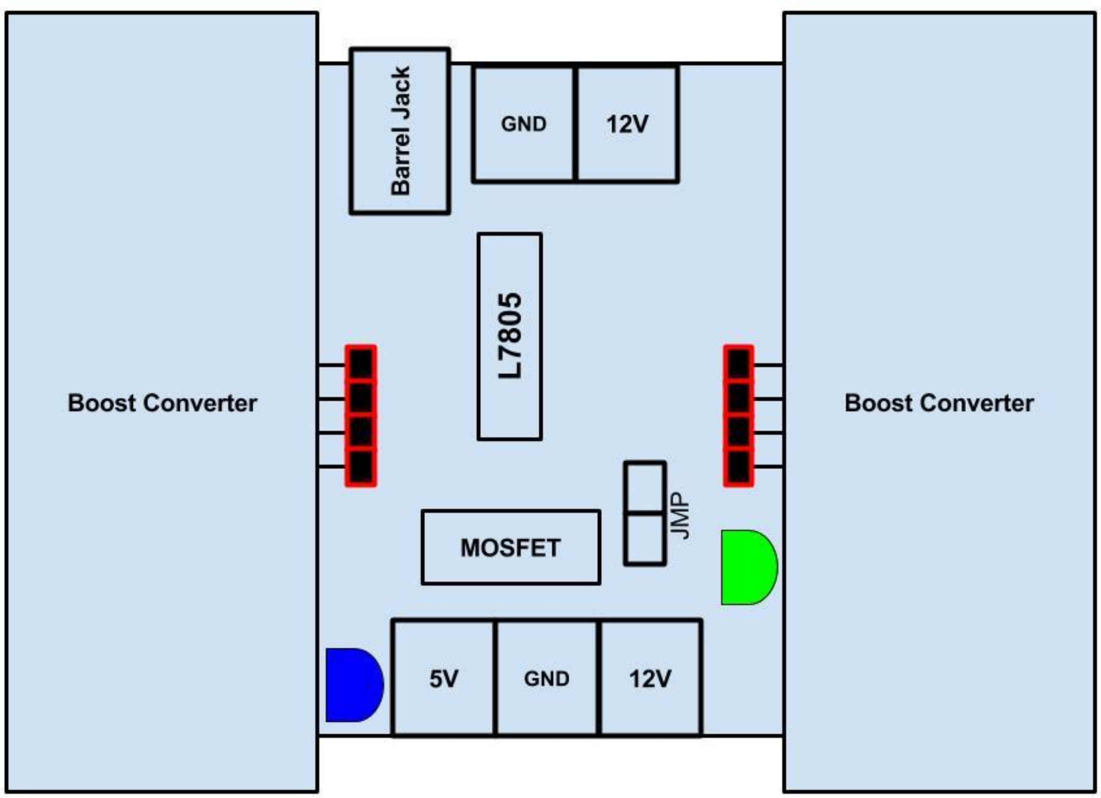
The ArduCam Mini 2MP has been set to capture images on jpeg format on successive burst to fill the buffer and then empty the buffer and restart the process. The camera is then set to send this images through the serial communication to the ESP8266 chip via the UART. The Images are then send to the web server and able to be viewed.



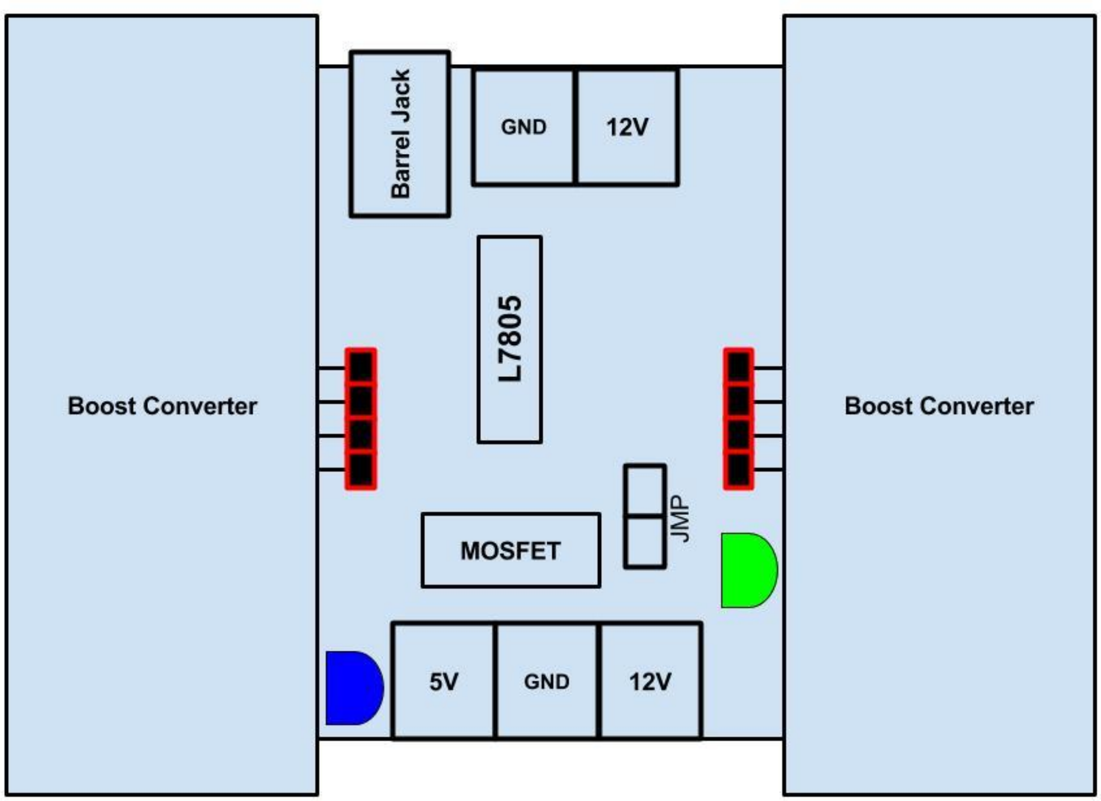
The Bluetooth communication will be implemented with a HC 06 Bluetooth chip capable of a transfer rate of 2 Mbits/second at 2.4GHz frequency. The HC 06 will be connected to the Arduino 2560 Mega via the serial ports. The computer will connect to the HC 06 and used to send the commands from MATLAB to the Arduino board.

Arduino Code

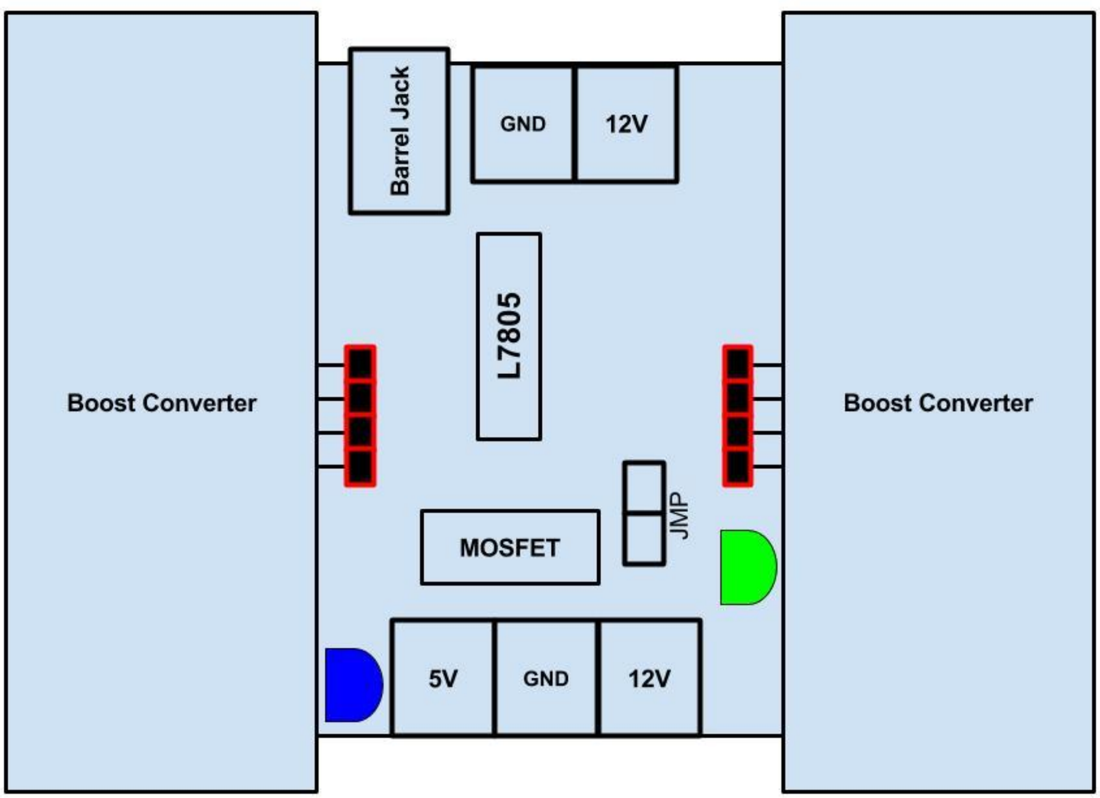
```
// This code is to create a streaming server which will
// interface with the MATLAB ipcam function
//it is a modified code from the sample code provided
// on the arducam tutorial.
// it only works with the Mini 2MP module and the Arducam ESP8266 uno board.
// and will only create a access point.
// Erik Cuevas
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <Wire.h>
#include <ArduCAM.h>
#include <SPI.h>
#include "memorysaver.h"
#if !(defined ESP8266 )
#error Please select the ArduCAM ESP8266 UNO board in the Tools/Board
```



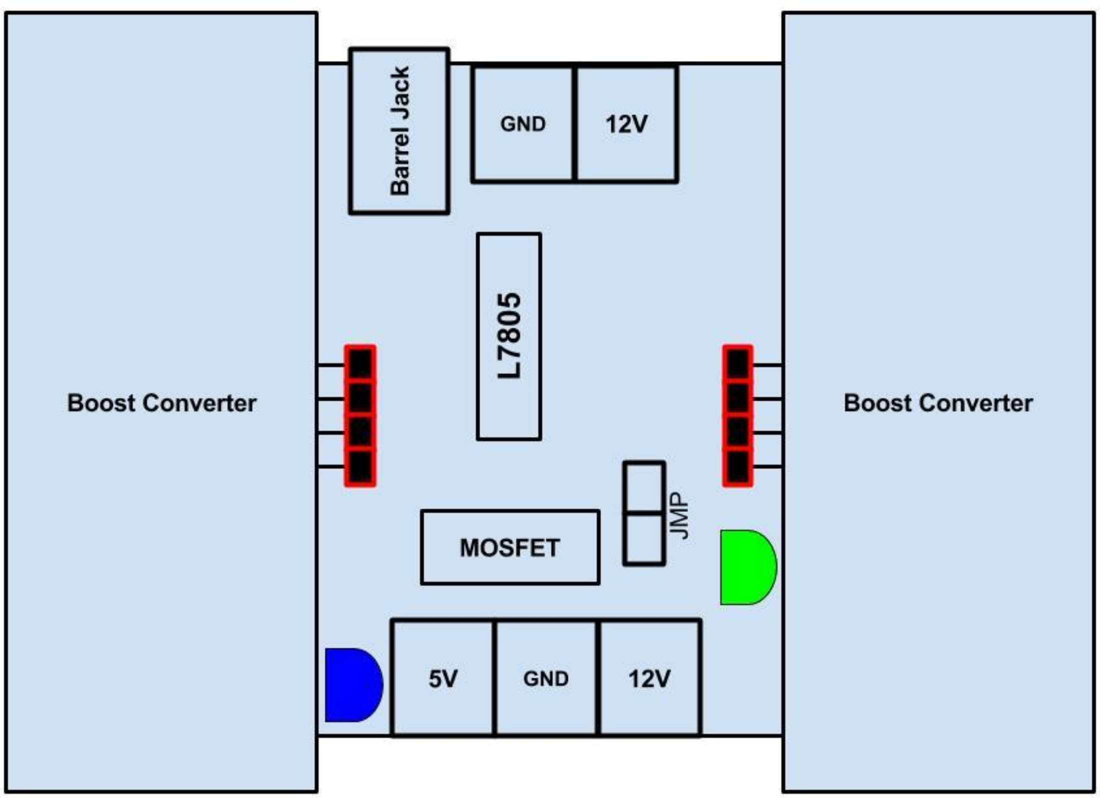
```
#endif

//This demo can only work on OV2640_MINI_2MP or ARDUCAM_SHIELD_V2 platform.
#if !(defined (OV2640_MINI_2MP))
#error Please select the hardware platform and camera module in the
../libraries/ArduCAM/memorysaver.h file
#endif
// set GPIO16 as the slave select :
const int CS = 16;


//AP mode configuration
//name your access point
const char *AP_ssid = "Team3";
//Default is no password.If you want to set password,put your password here
const char *AP_password = "";

static const size_t bufferSize = 4096;
static uint8_t buffer[bufferSize] = {0xFF};
uint8_t temp = 0, temp_last = 0;
int i = 0;
bool is_header = false;

ESP8266WebServer server(80);
#if defined (OV2640_MINI_2MP) || defined (OV2640_CAM)
  ArduCAM myCAM(OV2640, CS);
#endif

//
void start_capture(){
```



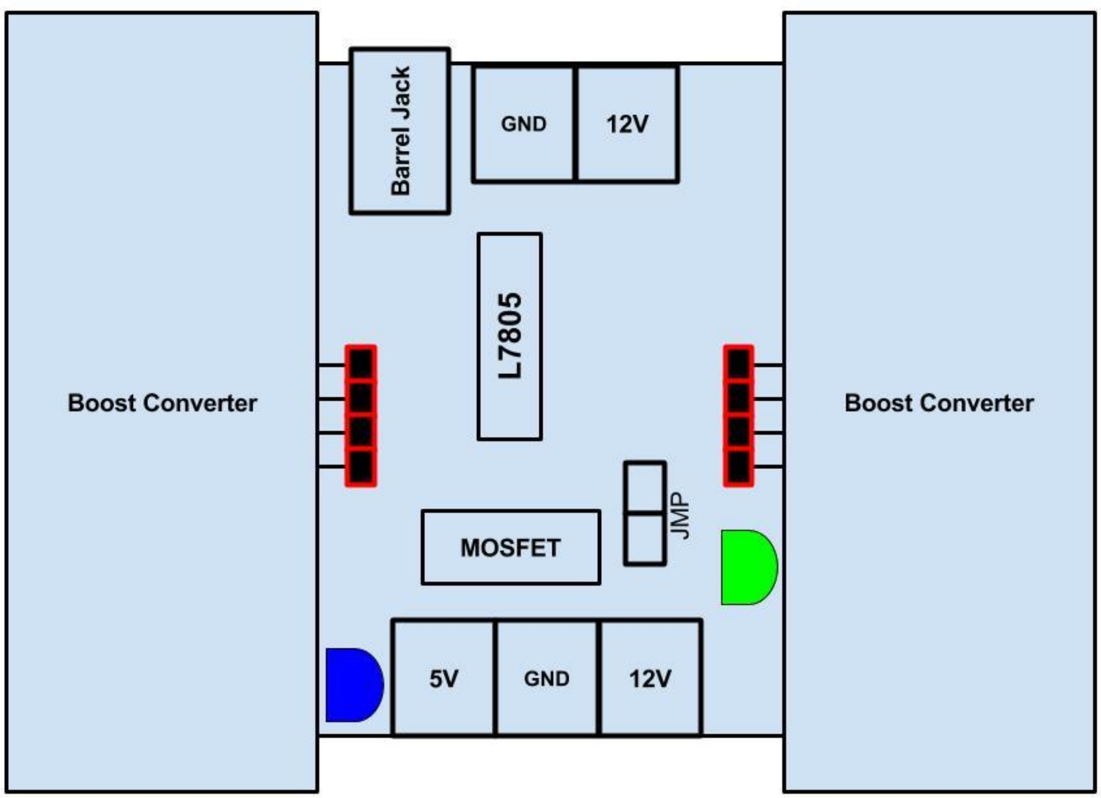
```
  myCAM.clear_fifo_flag();
  myCAM.start_capture();
}

void camCapture(ArduCAM myCAM){
WiFiClient client = server.client();
uint32_t len  = myCAM.read_fifo_length();
if (len >= MAX_FIFO_SIZE) //8M
{
  Serial.println(F("Over size."));
}
if (len == 0 ) //0 kb
{
  Serial.println(F("Size is 0."));
}
myCAM.CS_LOW();
myCAM.set_fifo_burst();
if (!client.connected()) return;
String response = "HTTP/1.1 200 OK\r\n";
response += "Content-Type: image/jpeg\r\n";
response += "Content-len: " + String(len) + "\r\n\r\n";
server.sendContent(response);
i = 0;
while ( len-- )
{
temp_last = temp;
temp =  SPI.transfer(0x00);
//Read JPEG data from FIFO
if ( (temp == 0xD9) && (temp_last == 0xFF) ) //If find the end ,break while,
{
```



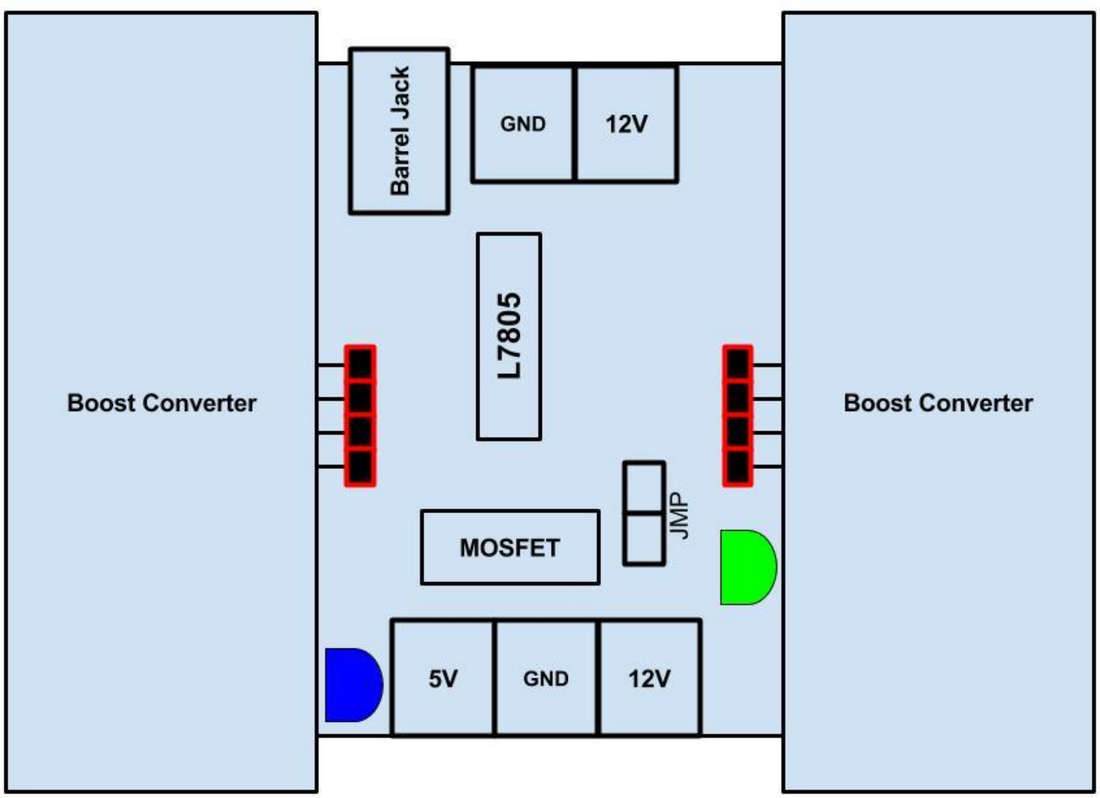
```
buffer[i++] = temp;  //save the last  0XD9
//Write the remain bytes in the buffer
if (!client.connected()) break;
client.write(&buffer[0], i);
is_header = false;
i = 0;
myCAM.CS_HIGH();
break;
}
if (is_header == true)
{
//Write image data to buffer if not full
if (i < bufferSize)
buffer[i++] = temp;
else
{
//Write bufferSize bytes image data to file
if (!client.connected()) break;
client.write(&buffer[0], bufferSize);
i = 0;
buffer[i++] = temp;
}
}
else if ((temp == 0xD8) & (temp_last == 0xFF))
{
is_header = true;
buffer[i++] = temp_last;
buffer[i++] = temp;
}
}
```



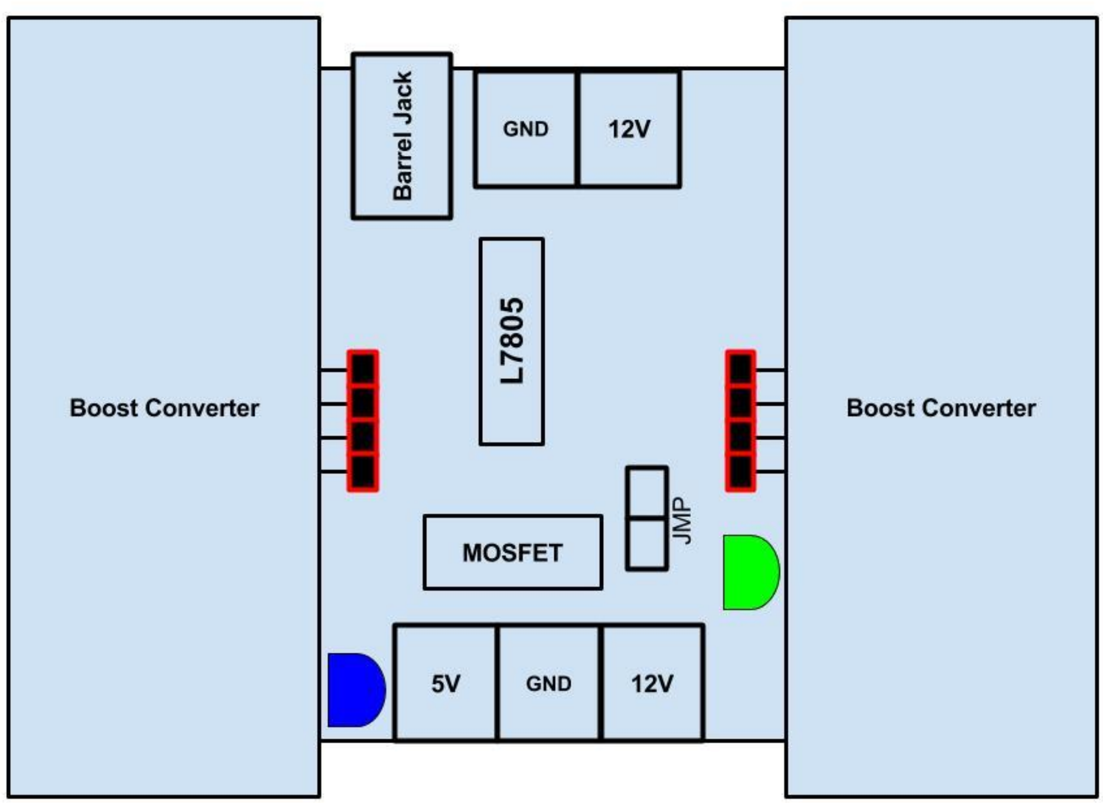
```
}

void serverStream(){
WiFiClient client = server.client();

String response = "HTTP/1.1 200 OK\r\n";
response += "Content-Type: multipart/x-mixed-replace; boundary=frame\r\n\r\n";
server.sendContent(response);

while (1){
start_capture();
while (!myCAM.get_bit(ARDUCHIP_TRIG, CAP_DONE_MASK));
size_t len = myCAM.read_fifo_length();
if (len >= MAX_FIFO_SIZE) //8M
{
Serial.println(F("Over size."));
continue;
}
if (len == 0 ) //0 kb
{
Serial.println(F("Size is 0."));
continue;
}
myCAM.CS_LOW();
myCAM.set_fifo_burst();
if (!client.connected()) break;
response = "--frame\r\n";
response += "Content-Type: image/jpeg\r\n\r\n";
server.sendContent(response);
```



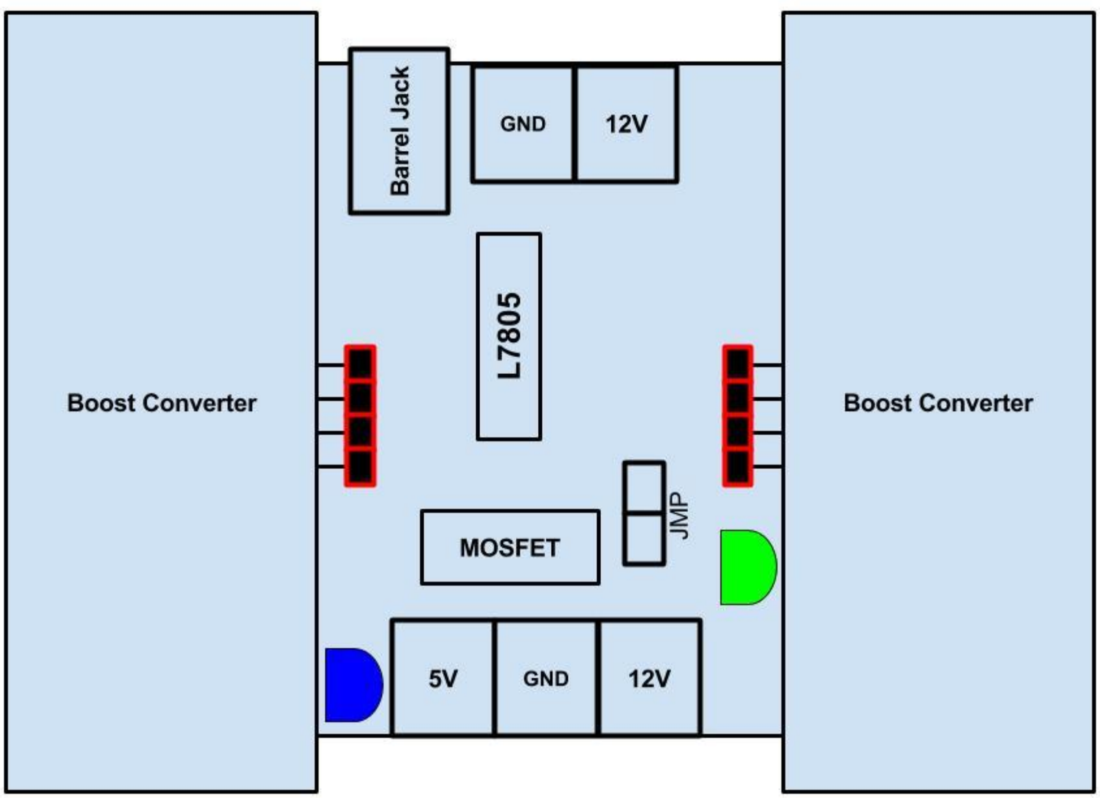
```
while ( len-- )
{
temp_last = temp;
temp =  SPI.transfer(0x00);

//Read JPEG data from FIFO
if ( (temp == 0xD9) && (temp_last == 0xFF) ) //If find the end ,break while,
{
buffer[i++] = temp;  //save the last  0XD9
//Write the remain bytes in the buffer
myCAM.CS_HIGH();;
if (!client.connected()) break;
client.write(&buffer[0], i);
is_header = false;
i = 0;
}
if (is_header == true)
{
//Write image data to buffer if not full
if (i < bufferSize)
buffer[i++] = temp;
else
{
//Write bufferSize bytes image data to file
myCAM.CS_HIGH();
if (!client.connected()) break;
client.write(&buffer[0], bufferSize);
i = 0;
buffer[i++] = temp;
myCAM.CS_LOW();
```



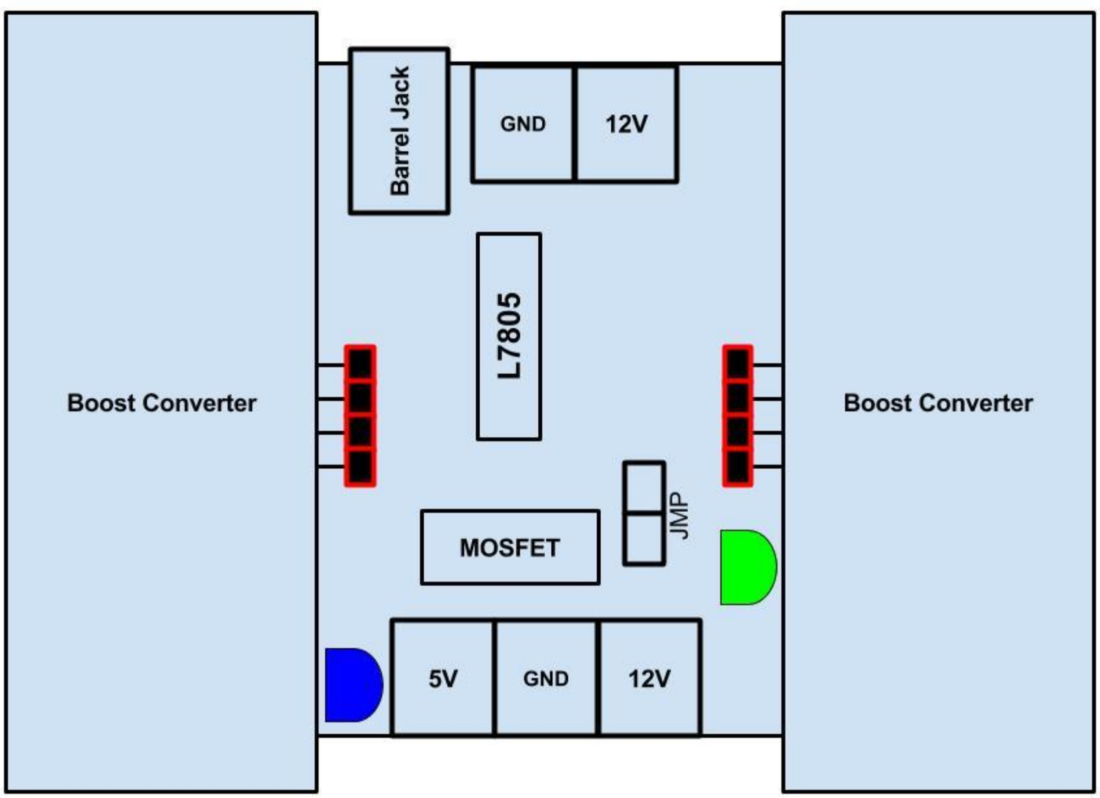
```
myCAM.set_fifo_burst();
}
}
else if ((temp == 0xD8) & (temp_last == 0xFF))
{
is_header = true;
buffer[i++] = temp_last;
buffer[i++] = temp;
}
}
if (!client.connected()) break;
}
}

void handleNotFound(){
String message = "Server is running!\n\n";
message += "URI: ";
message += server.uri();
message += "\nMethod: ";
message += (server.method() == HTTP_GET)?"GET":"POST";
message += "\nArguments: ";
message += server.args();
message += "\n";
server.send(200, "text/plain", message);

if (server.hasArg("ql")){
int ql = server.arg("ql").toInt();
#if defined (OV2640_MINI_2MP) || defined (OV2640_CAM)
myCAM.OV2640_set_JPEG_size(ql);
#endif
```



```
delay(1000);
Serial.println("QL change to: " + server.arg("ql"));
}
}

void setup() {
uint8_t vid, pid;
uint8_t temp;
#if defined(__SAM3X8E__)
Wire1.begin();
#else
Wire.begin();
#endif
Serial.begin(115200);
Serial.println(F("ArduCAM Start!"));

// set the CS as an output:
pinMode(CS, OUTPUT);

// initialize SPI:
SPI.begin();
SPI.setFrequency(4000000); //4MHz

//Check if the ArduCAM SPI bus is OK
myCAM.write_reg(ARDUCHIP_TEST1, 0x55);
temp = myCAM.read_reg(ARDUCHIP_TEST1);
if (temp != 0x55){
Serial.println(F("SPI1 interface Error!"));
while(1);
}
```



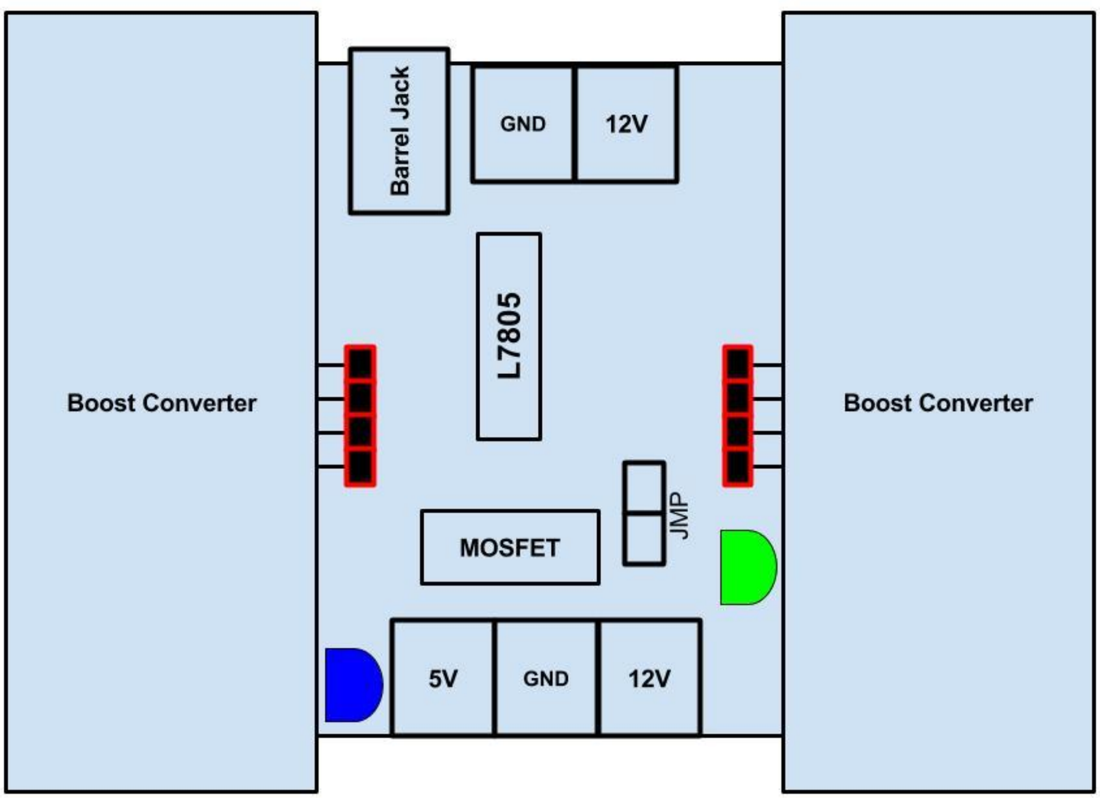
```
#if defined (OV2640_MINI_2MP) || defined (OV2640_CAM)
//Check if the camera module type is OV2640
myCAM.wrSensorReg8_8(0xff, 0x01);
myCAM.rdSensorReg8_8(OV2640_CHIPID_HIGH, &vid);
myCAM.rdSensorReg8_8(OV2640_CHIPID_LOW, &pid);
if ((vid != 0x26 ) && (( pid != 0x41 ) || ( pid != 0x42 )))
Serial.println(F("Can't find OV2640 module!"));
else
Serial.println(F("OV2640 detected."));
#endif


//Change to JPEG capture mode and initialize the OV2640 module
myCAM.set_format(JPEG);
myCAM.InitCAM();
#if defined (OV2640_MINI_2MP) || defined (OV2640_CAM)
myCAM.OV2640_set_JPEG_size(OV2640_320x240);
#endif

myCAM.clear_fifo_flag();
Serial.println();
Serial.println();
Serial.print(F("Share AP: "));
Serial.println(AP_ssid);
Serial.print(F("The password is: "));
Serial.println(AP_password);

WiFi.mode(WIFI_AP);
WiFi.softAP(AP_ssid, AP_password);
Serial.println("");
```



```
Serial.println(WiFi.softAPIP());


// Start the server
server.on("/video.mjpeg", HTTP_GET, serverStream);
server.onNotFound(handleNotFound);
server.begin();
Serial.println(F("Server started"));
}

void loop() {
server.handleClient();
}
```

Weapon Design and Implementation

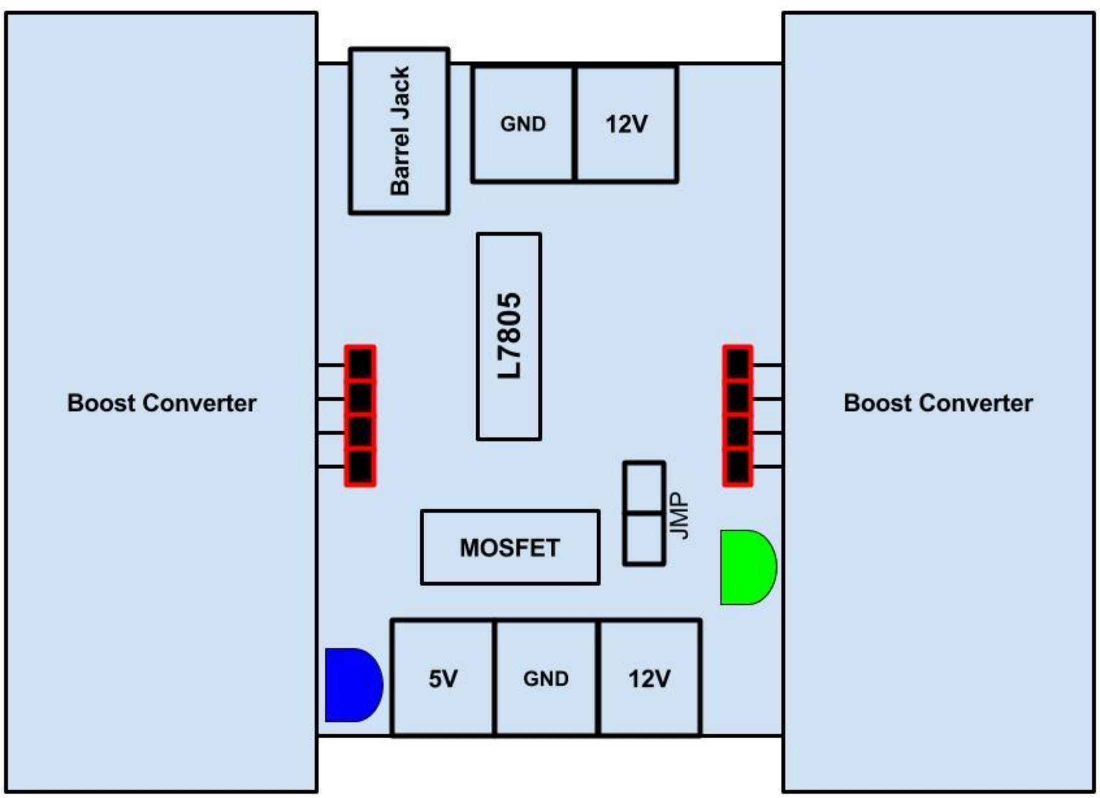
The A.N.T. is fitted with an air powered cannon that is capable of firing a single Nerf dart over 15 ft. The current design implements the use of flexible polymer tubing to store pressurized air, which is allowed to flow, in short bursts, through a DC controlled solenoid valve. The outlet of the solenoid valve is input to a narrow flexible rubber tube, which will attach to the backside of the the cannon barrel, adjacently mounted to the target recognition camera on a servo.



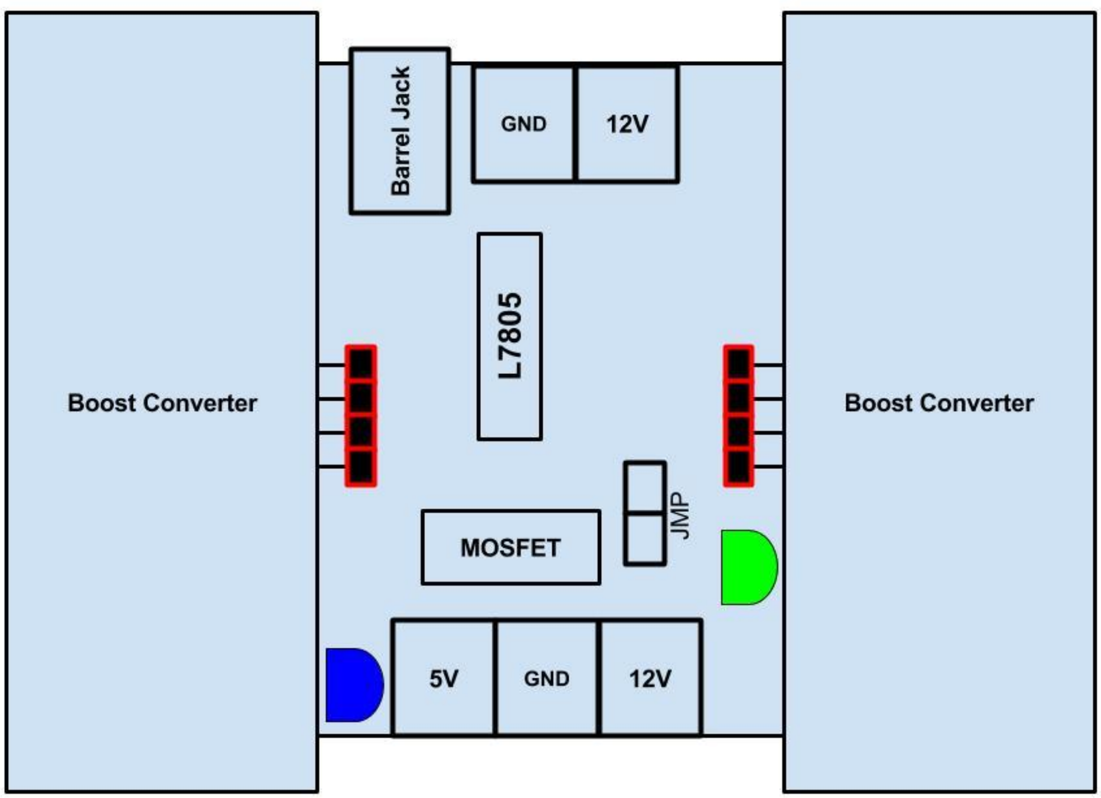
The firing of the cannon is a two step process to act as a safety measure. The main loop of code will autonomously navigate the A.N.T. through a maze as well as identify and lock onto a target. Once image processing is complete and the cannon is locked onto the target, the code will ask to 'arm' the weapon. User input will provide the system with a decision, yes or no. Once the weapon is armed, the user has the ability to input a fire command to activate the solenoid. This method also removes firing of the weapon from the A.N.T.'s decision making process, thus preventing unintentional attacks. The MOSFET used to drive the solenoid has the gate pulled to ground via a 10k resistor to further prevent accidental discharge of the cannon during operation. There is a green LED also attached to the gate of the MOSFET to indicate activation of the gate pin.

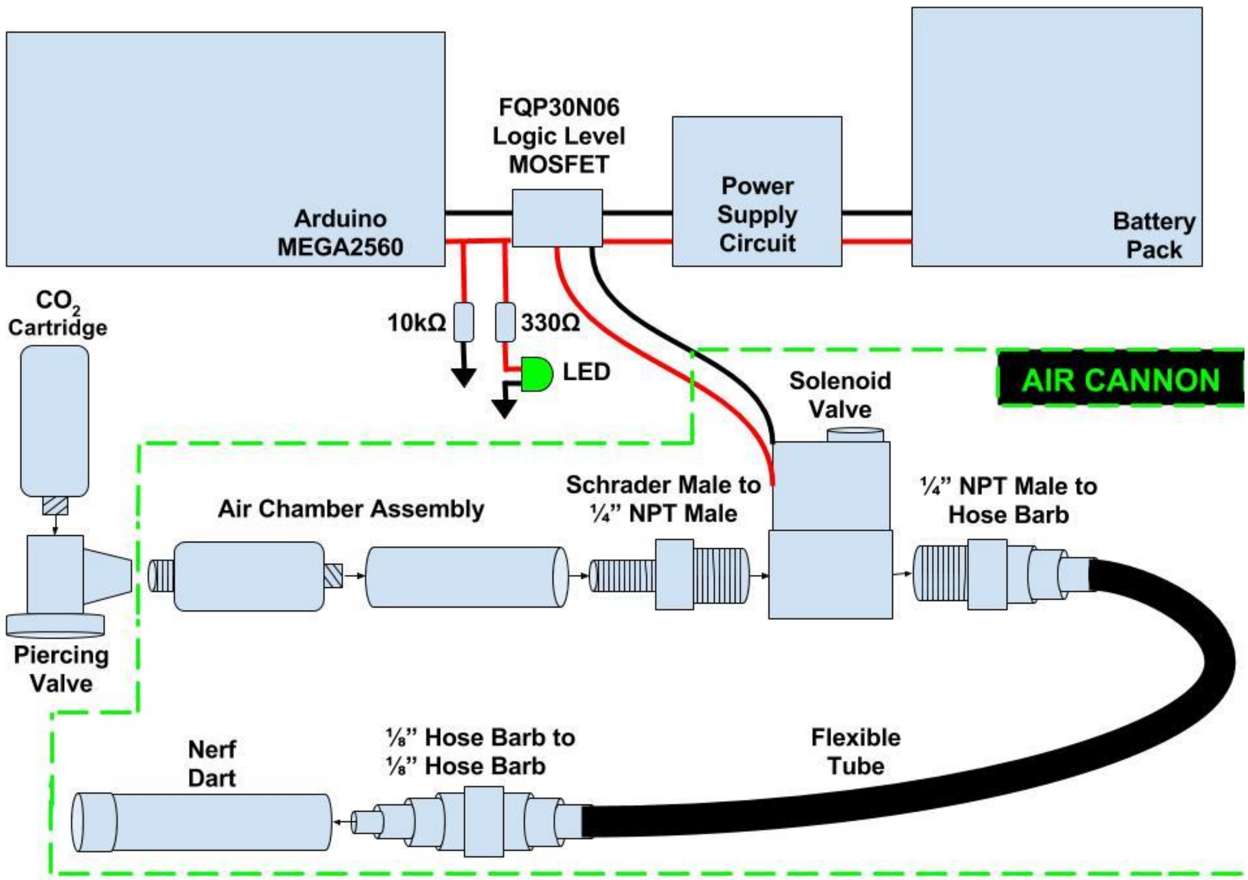
Air Cannon Hardware

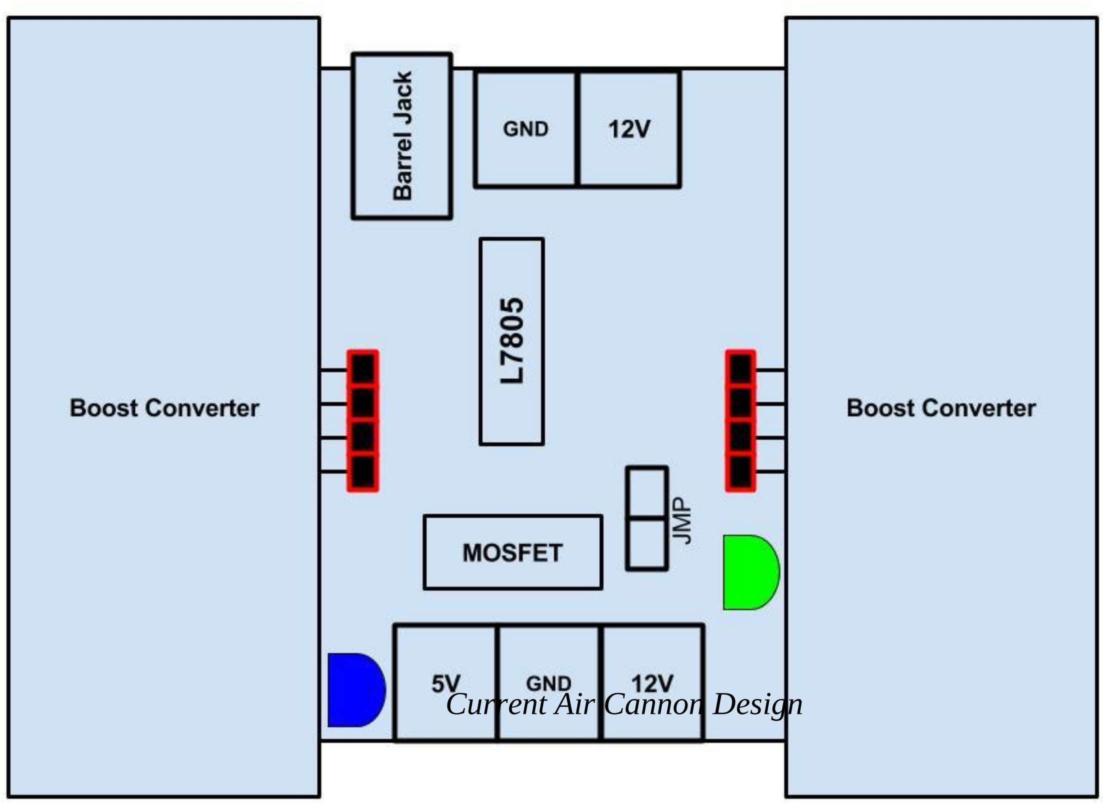
The diagram below shows the required components and configuration for the air cannon. Inside of the dotted box we have the main air cannon assembly. Starting on the left side of the air



cannon, the air chamber assembly is the vessel in which air is pressurized to between 30 and 50 psi. This air chamber is composed of a schrader valve mounted in an empty CO_2 cartridge which is attached to one end of the polymer tubing using a hose clamp. The other end of the polymer tubing is then attached to the solenoid valve using ¼" NPT fittings. The solenoid valve is an N.C. 12V DC device capable of operating at 1Mpa(~145 psi), which, when energized, allows the pressurized air to flow through. The outlet of this valve is fed into a smaller flexible rubber tube that is then routed to a hose barb(cannon barrel) which is mounted on the targeting camera servo assembly. Operation of the cannon begins by charging the air chamber assembly using a bicycle tire pump. The polymer tubing should be rigid when properly charged. Once the cannon is ready to be fired, the arduino will send a 'HIGH' logic signal to the MOSFET, applying a 12V drop across the solenoid leads, activating the valve.



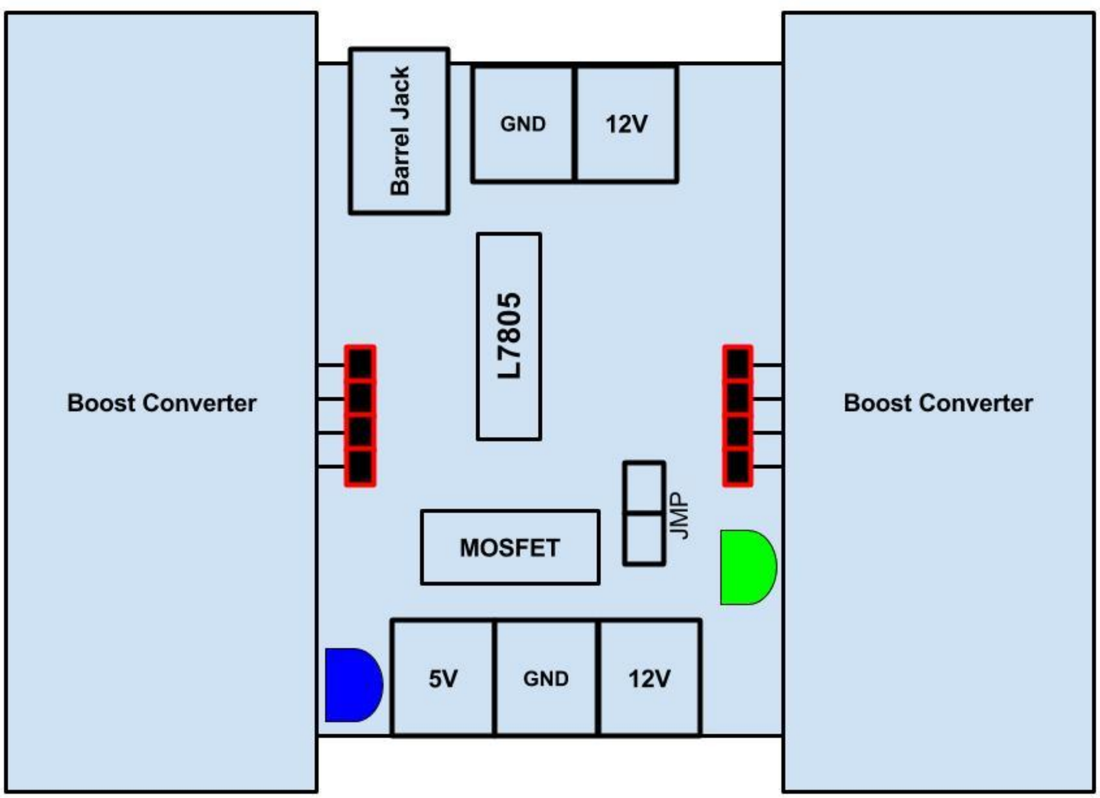


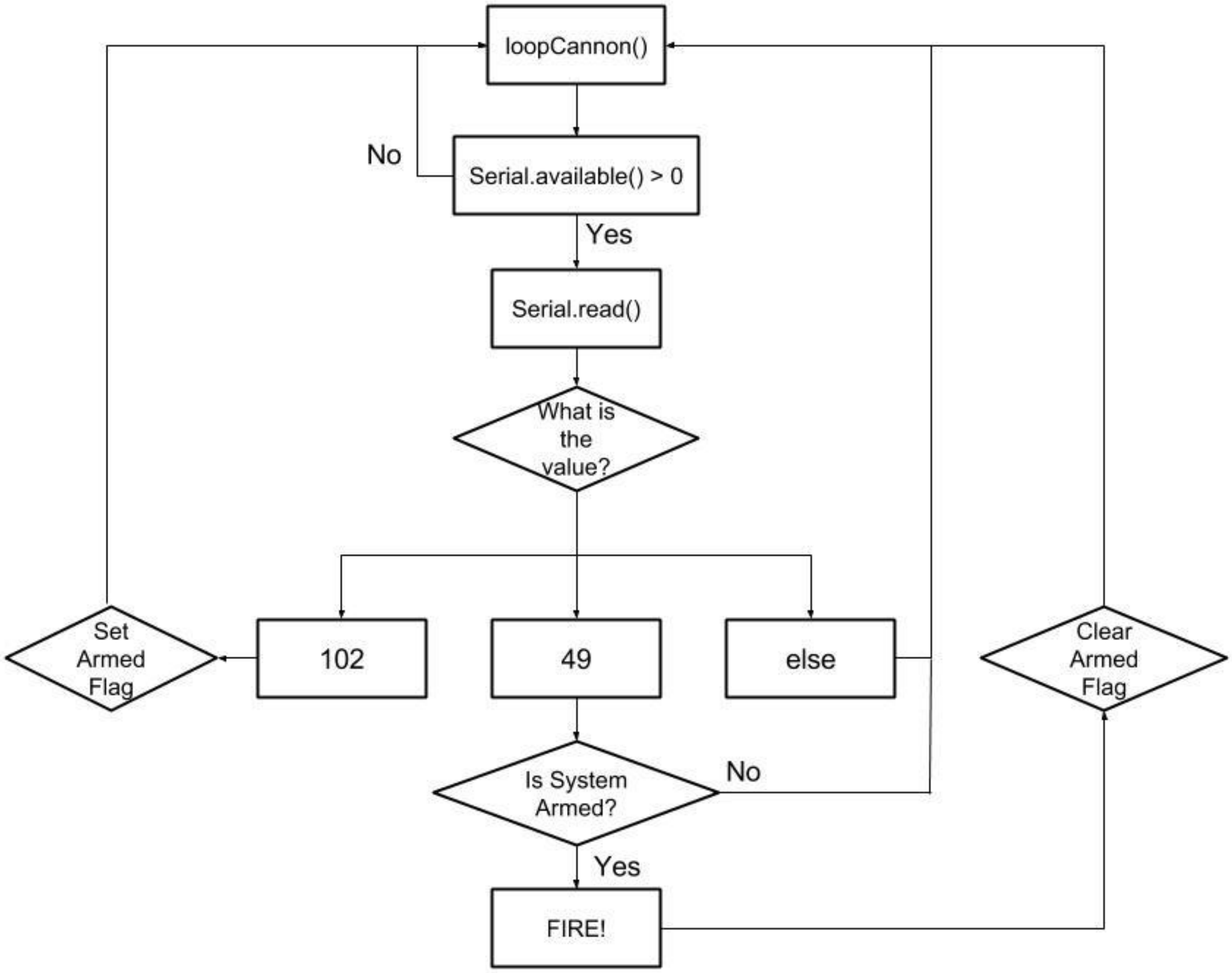


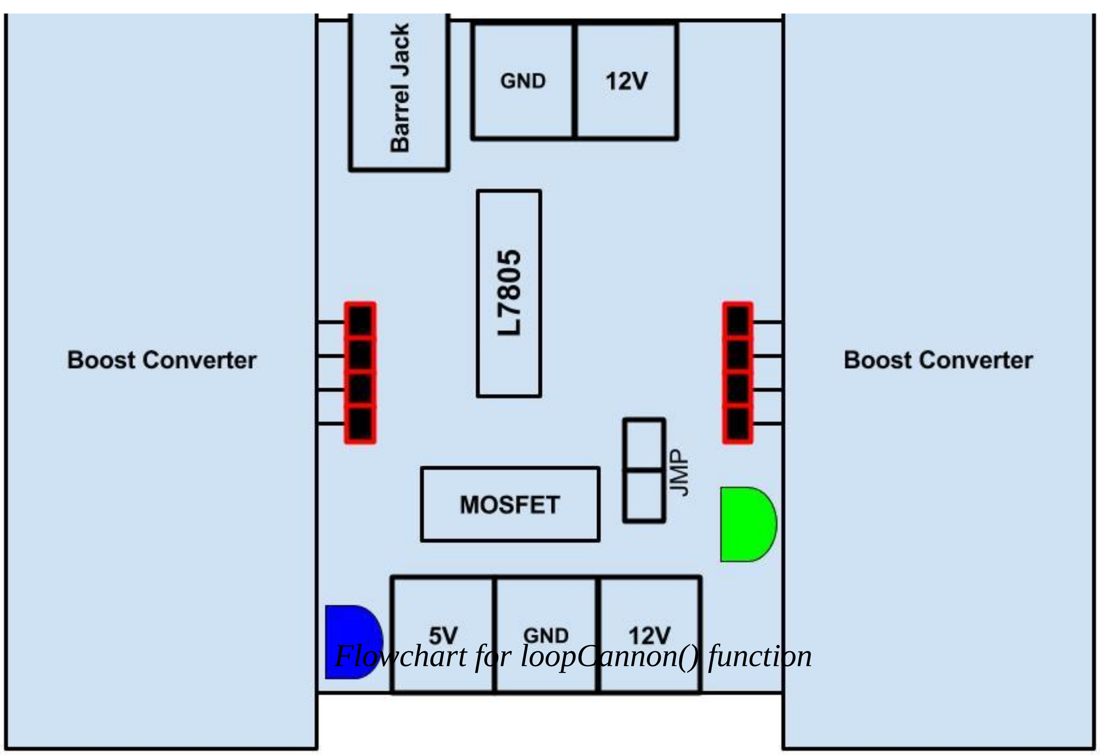
Current Air Cannon Design

Air Cannon Code

In order to make the air cannon code easily integratable, it was written as two separate, passive functions that directly operate based on data in the serial bus. For ease of use, the script file, airCannon.ino should be added as a new page in the main script. The two functions in the script are setupCannon() and loopCannon(). The function, setupCannon, initializes the pins and variables needed to use the cannon and should be called at the beginning of setup(). The function, loopCannon() is placed within a control loop that is only called when the system has identified that it has locked onto a target. Once loopCannon() is active, a Serial.print() of ASCII character ‘F’ will cause the weapon to arm, and a Serial.print() of ASCII character ‘1’ will cause the weapon to fire. The loopCannon flowchart and the code are provided below.



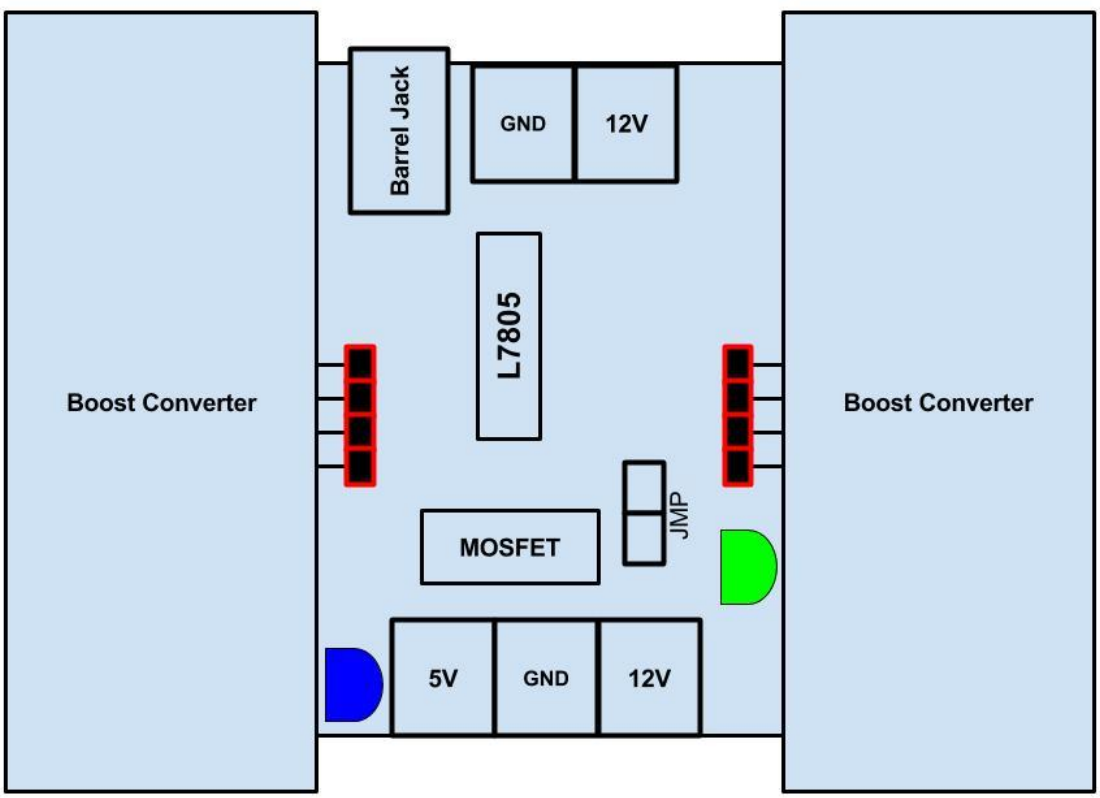




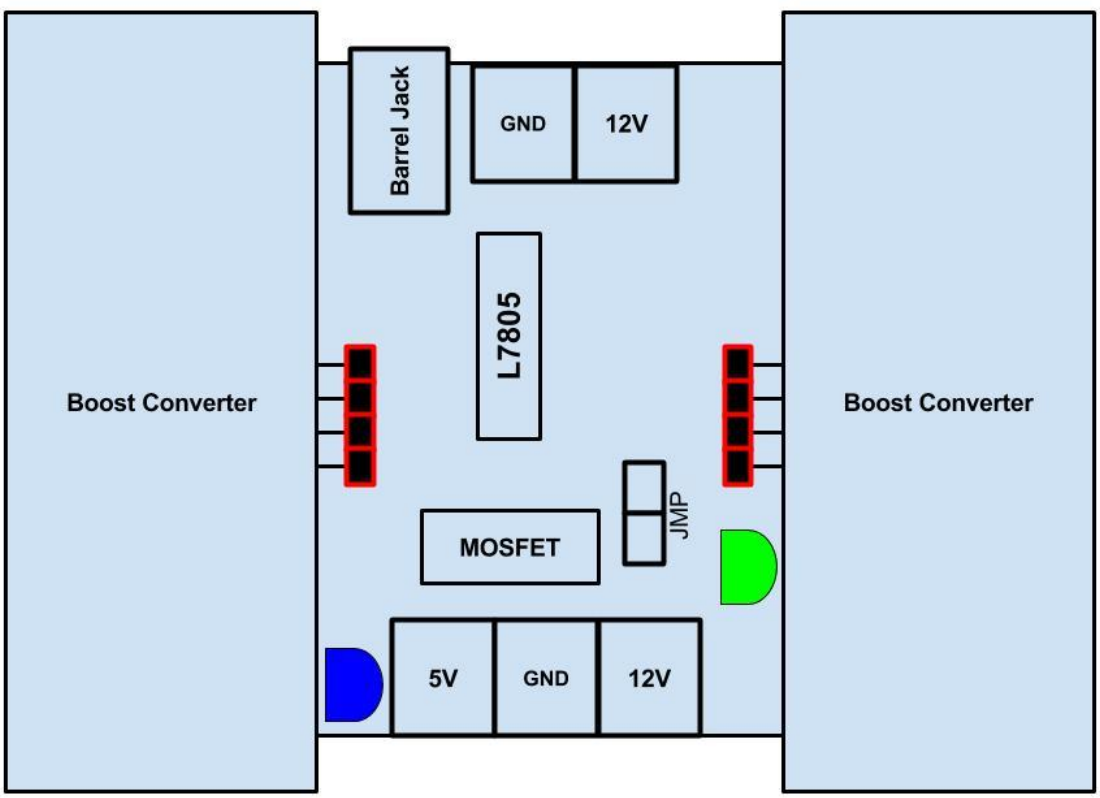
Flowchart for loopCannon() function

The following is the code for airCannon.ino

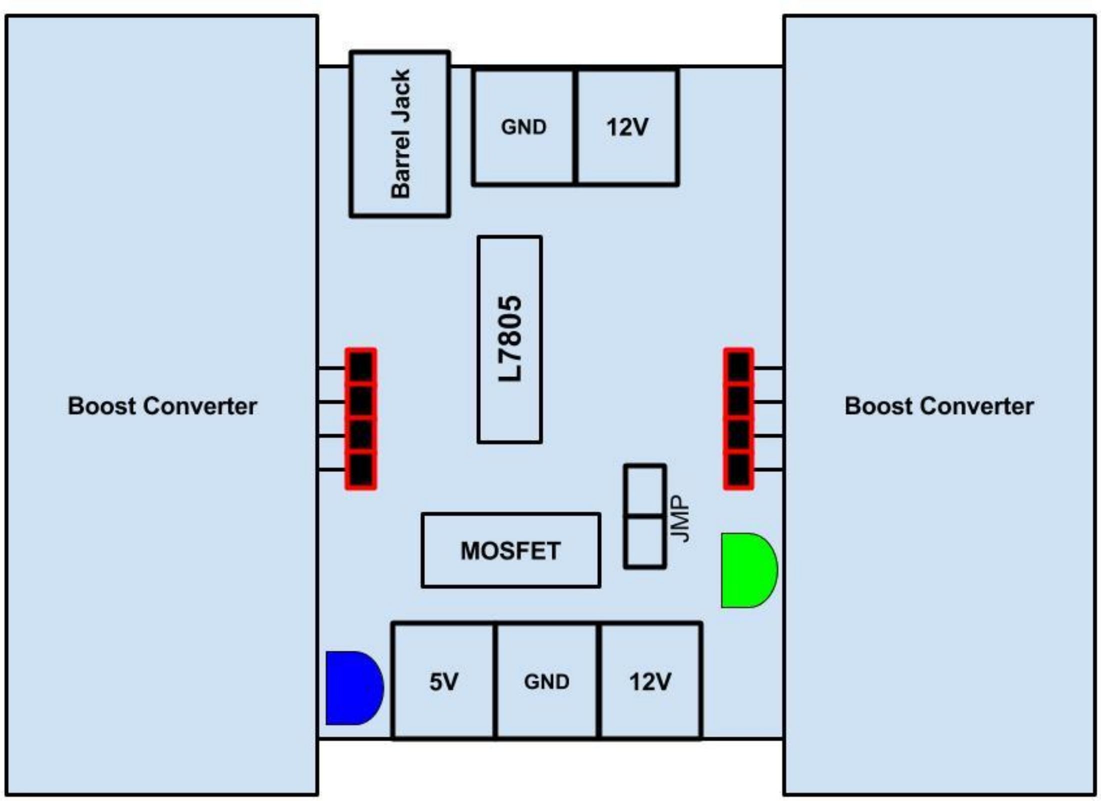
```
/***Pin Definitions***/

#define firePin 12              //This pin connects to gate of MOSFET
#define fireButton 21           //This pin connects to onboard manual fire button
#define armButton 20            //This pin connects to onboard manual arming button
#define speakerPin 11           //This pin connects to small speaker
#define armedIndicator A0       //This pin connects to an LED to indicate weapon status

#define whichSerial Serial      //Choose serial port for comms (Serial1, Serial2, ...)
#define armValue 102            //Choose value to send to arm weapon
#define fireValue 49            //Choose value to send to fire weapon
#define fireDelay 100           //Choose delay between open and closed airflow
/***Variables***/

int serialIn = 0;               //This variable carries user entered serial commands
/****************************************
Currently, the commands recognized are:

                    'F' = Arm Weapon
                    '1' = Fire Weapon
                    else = Do Nothing
*****************************************/
volatile byte weaponArmed = 0; //This variable is a weapon status flag


/////////////////////////////////////////////////////////
```



```
// Comment this section if implementing code with system
// Uncomment this section if testing code separately
/*
void setup(){
  setupCannon();
}

void loop(){
  loopCannon();
}
*/
////////////////////////////////////////////////////////

void setupCannon() {
  //Initialize i/o pins
  pinMode(firePin, OUTPUT);
  pinMode(fireButton, INPUT);
  pinMode(armButton, INPUT);
  pinMode(speakerPin, OUTPUT);
  //Begin serial comms
  whichSerial.begin(9600);
  //Attach interrupts to functions
  attachInterrupt(digitalPinToInterrupt(fireButton), fireWeapon, LOW);
  attachInterrupt(digitalPinToInterrupt(armButton), armWeapon, LOW);
}

void loopCannon() {
  if(whichSerial.available() > 0){
    serialIn = whichSerial.read();
    serialTriggers(serialIn);
```



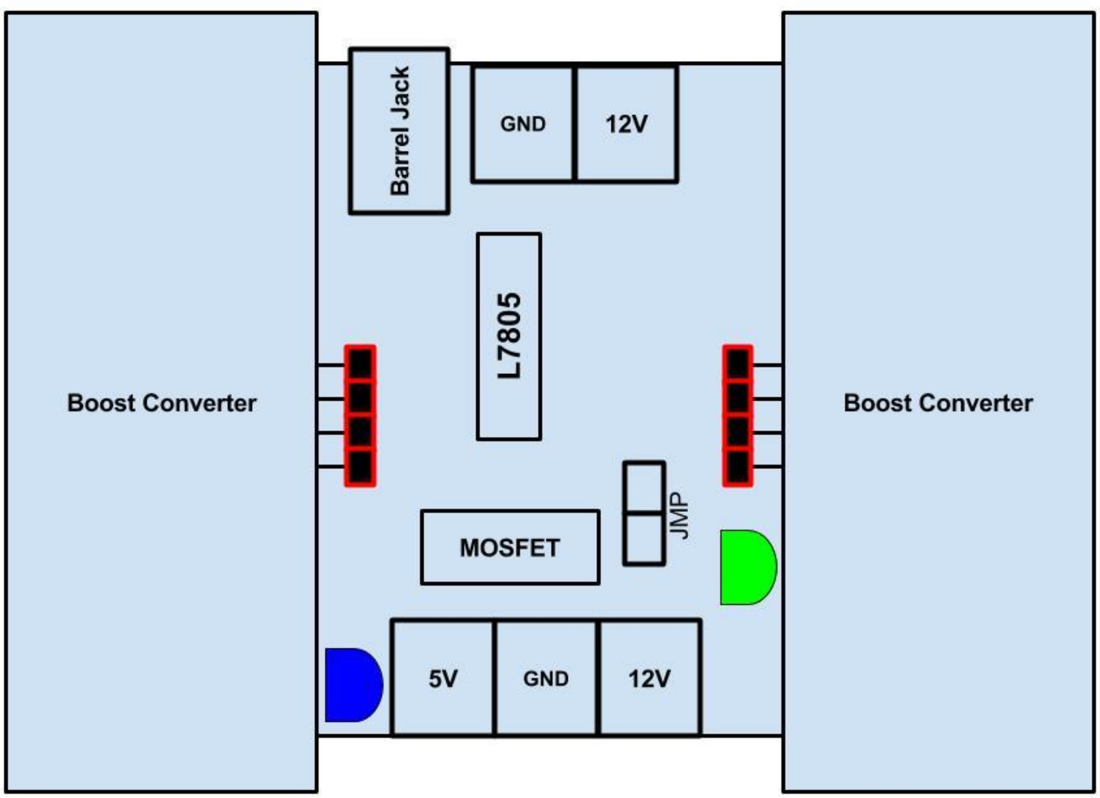
```
    }
  }

  void serialTriggers(int ser){  //Process serial commands
    if(ser == armValue){         //If input is armValue, call armWeapon
      armWeapon();
    }
    else if(ser == fireValue){   //if input is fireValue, call fireWeapon
      fireWeapon();
    }
    else{                        //If input is anything else, call doNothing
      doNothing();
    }
  }

  void armWeapon(){                          //This function arms the weapon
    while(digitalRead(armButton) == LOW){   //Debounce
      delay(50);
    }
    if(weaponArmed != 1){                    //Check if weapon is not already armed
      weaponArmed = 1;                       //If not, set arming flag
      whichSerial.println("ARMED");          //Print status to serial window
      analogWrite(armedIndicator, 128);      //Illuminate arm status LED
      tone(speakerPin, 3780, 180);           //Play arming tone
    }
  }

  void fireWeapon(){                          //This function fires the weapon
    while(digitalRead(fireButton) == LOW){   //Debounce
      delay(50);
```



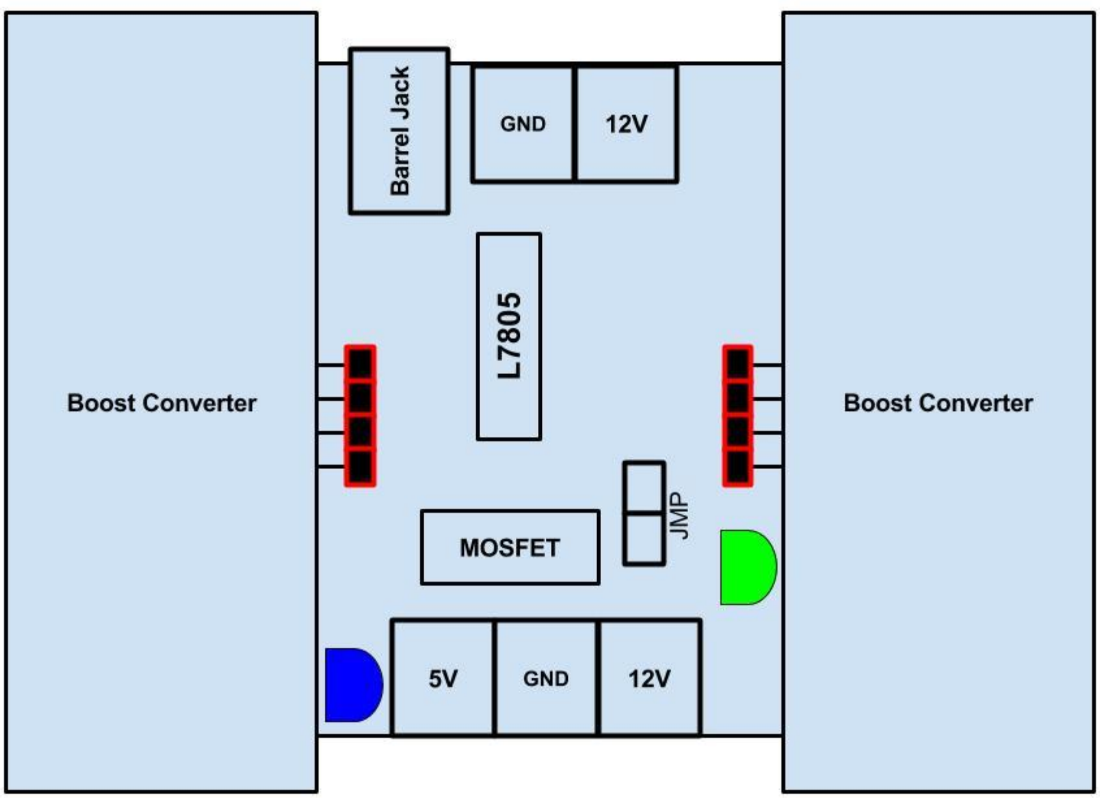
```
  }
  if(weaponArmed == 1){                    //Check if weapon is armed
    weaponArmed = 0;                       //If armed, unset arming flag
    digitalWrite(firePin, HIGH);           //Set firing pin to HIGH(enable air flow)
    delay(fireDelay);                            //Wait
    digitalWrite(firePin, LOW);            //Set firing pin to LOW(disable air flow)
    analogWrite(armedIndicator, 0);        //Turn off arm status LED
    whichSerial.println("DEPLOYED");       //Print status to serial window
    tone(speakerPin, 1890, 1000);          //Play firing complete tone
  }
}

void doNothing(){           //This function only prints "Do Nothing" to serial window
      // can be used to execute other actions
      // when serial input is not as expected.
      // whichSerial.println("Do Nothing");
}
```

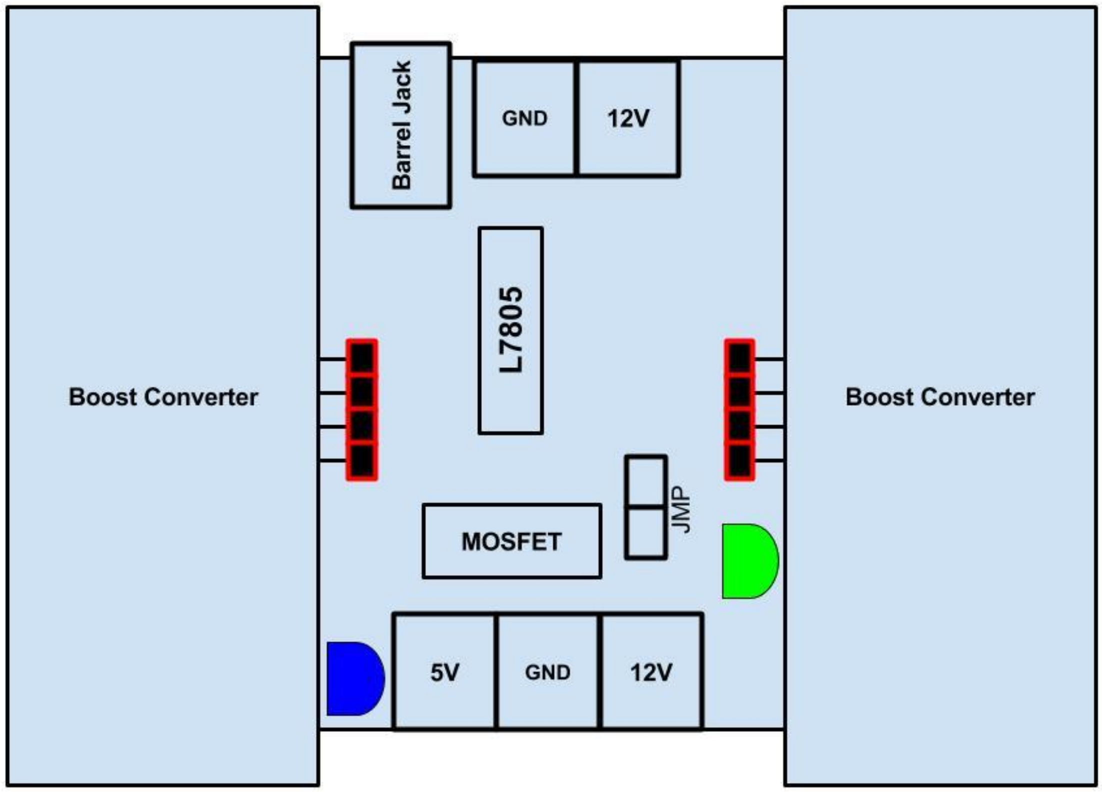
Power Supply

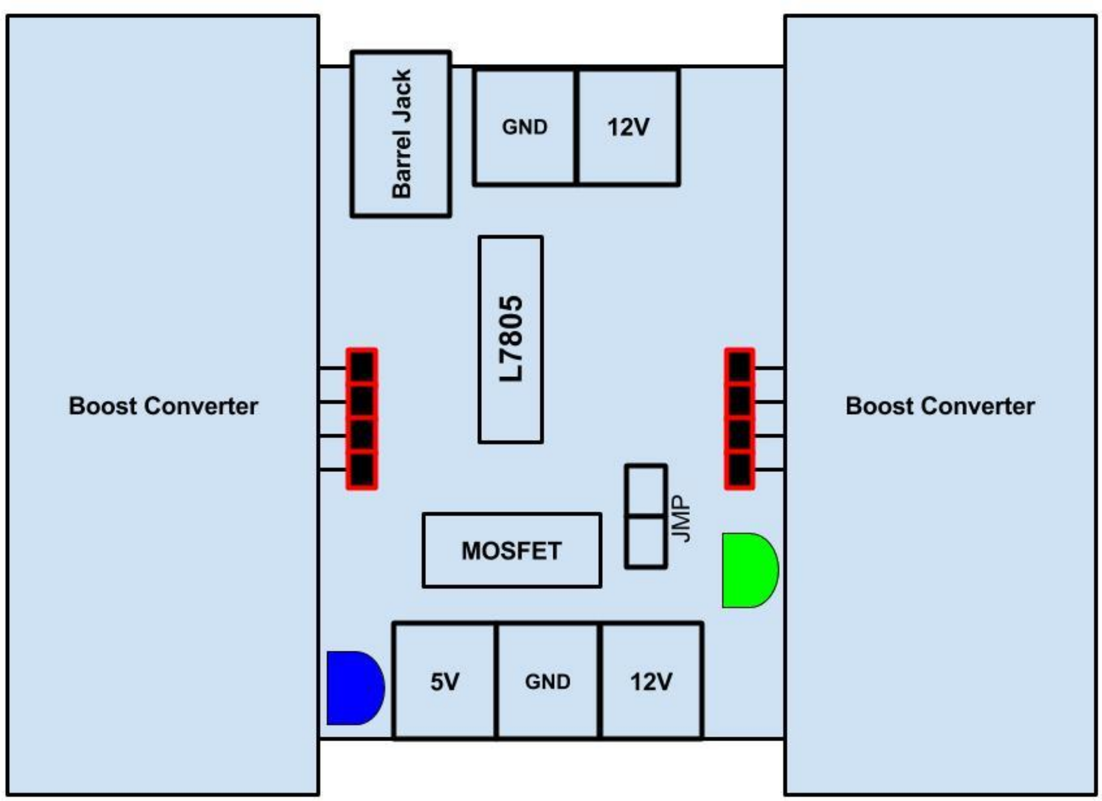
The A.N.T.'s power is provided by three parallel, 4.2V 18650 Li-ion cells capable of providing 2200mAh a piece. With a maximum safe discharge of 40A, these batteries are more than capable of providing the power necessary to run the tank motors as well as all of the peripherals. The battery pack is input to a pair of boost converters to help efficiently convert the

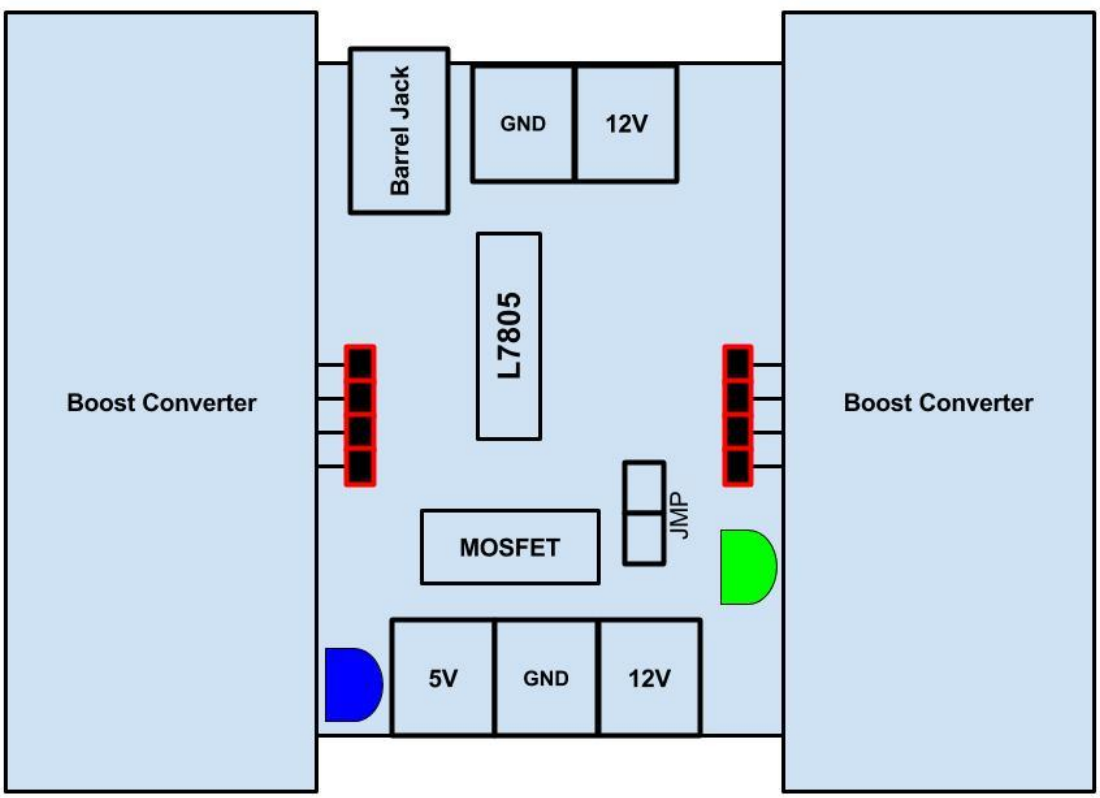


source to useable voltages. One boost converter is used to supply 12V, 3A to the L298N in order to control the motors and the air cannon solenoid. The other is boost converter is used to power the arduino and all other peripherals. The power supply board was designed in modular way to allow for flexibility in future adjustments. In the diagram below, the blocks labeled as boost converters can be replaced with other DC-DC converter topologies such as buck, SEPIC, or Cuk converters via the 4 pin headers(highlighted in red) on the pcb. The 5V regulator and MOSFET can each also be replaced with other IC's with the same function and pinout.




Barrel Jack
GND
12V
L7805
Boost Converter
Boost Converter
JMP
MOSFET
5V
GND
12V


Barrel Jack
GND
12V
L7805
Boost Converter
Boost Converter
JMP
MOSFET
5V
GND
12V


Boost Converter
Barrel Jack
GND
12V
L7805
JMP
MOSFET
5V
GND
12V
Boost Converter