ECE 402

EE DESIGN PROJECTS

Spring 2014

**FINAL DESIGN REVIEW MANUAL**

Autonomous Arm

by

Team Rock Rover

|  |  |  |
| --- | --- | --- |
| Jianxin Chen | Software | chen395 |
| Jiyuan Zhao | Sensors | zhao89 |
| Evan Stockrahm | Servo & Arm | estockra |
| Minki Chang | Power Supply | chang71 |
|  |  |  |

Table of Contents

[1. System Summary 5](#_Toc385577539)

[1.1. System Block Diagram 5](#_Toc385577540)

[1.2. Setup Instructions 6](#_Toc385577541)

[1.3. Operating Instructions 7](#_Toc385577542)

[2. Subsystem Chapter 8](#_Toc385577543)

[2.1. Software 8](#_Toc385577544)

[2.1.1. Technical Overview of Subsystem 8](#_Toc385577545)

[Communication with Robotic Arm 8](#_Toc385577546)

[Connecting to Ethernet 8](#_Toc385577547)

[2.1.2. Block Diagram 9](#_Toc385577548)

[2.1.3. Flow Chart 10](#_Toc385577549)

[2.1.4. Manufacturing Tests and Equipment List 11](#_Toc385577550)

[2.1.5. Parts Costs and Power Requirements 12](#_Toc385577551)

[2.2. Sensors 13](#_Toc385577552)

[2.2.1. Technical Overview of Subsystem 13](#_Toc385577553)

[2.2.2. Block Diagram 14](#_Toc385577554)

[2.2.3. Flow Chart 15](#_Toc385577555)

[2.2.4. Schematics 16](#_Toc385577556)

[2.2.5. Manufacturing Tests and Equipment List 17](#_Toc385577557)

[2.2.6. Parts Costs and Power Requirements 18](#_Toc385577558)

[2.3. Servos and Arm 19](#_Toc385577559)

[2.3.1. Technical Overview of Subsystem 19](#_Toc385577560)

[2.3.2. Block Diagram 20](#_Toc385577561)

[2.3.3. Flow Chart 21](#_Toc385577562)

[2.3.4. Manufacturing tests and Equipment list 22](#_Toc385577563)

[2.3.5. Parts Costs and Power Requirements 23](#_Toc385577564)

[2.4. Power Supply 24](#_Toc385577565)

[2.4.1. Power Supply Overview 24](#_Toc385577566)

[2.4.2. Block Diagram 25](#_Toc385577567)

[2.4.3. Schematic 26](#_Toc385577568)

[2.4.4. Manufacturing Test and Equipment List 27](#_Toc385577569)

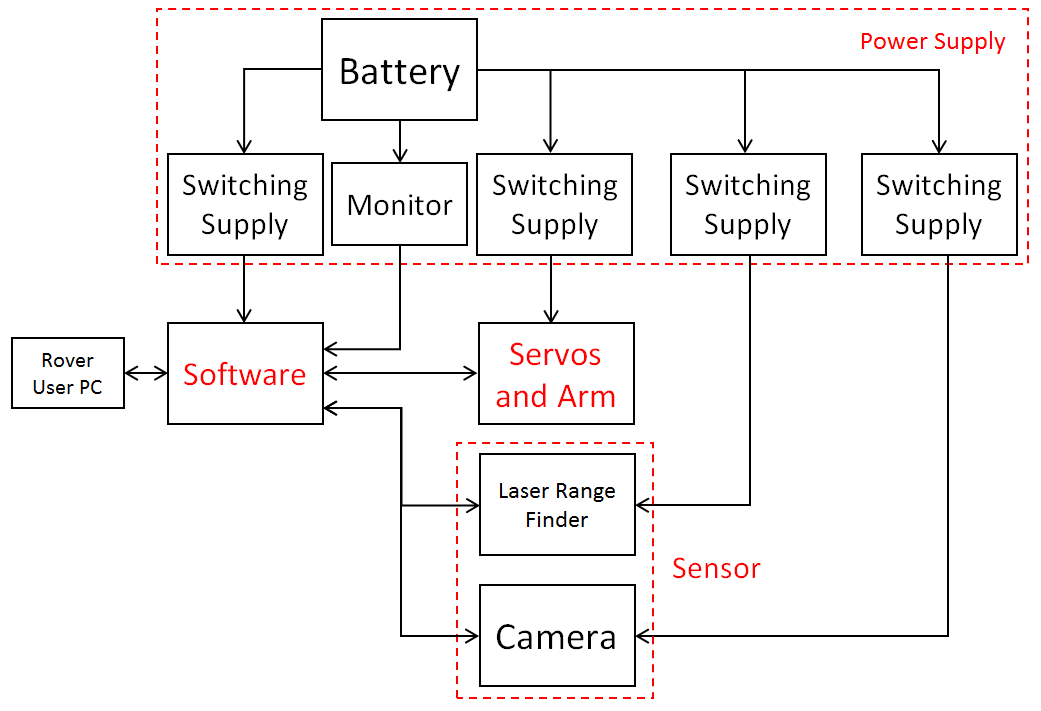
[2.4.5. Parts Costs and Power Requirements 28](#_Toc385577570)

[Appendix A. Example Appendix 29](#_Toc385577571)

[A.1. Servo and Arm Subsystem BotBoarduino Code 29](#_Toc385577572)

# System Summary

## System Block Diagram



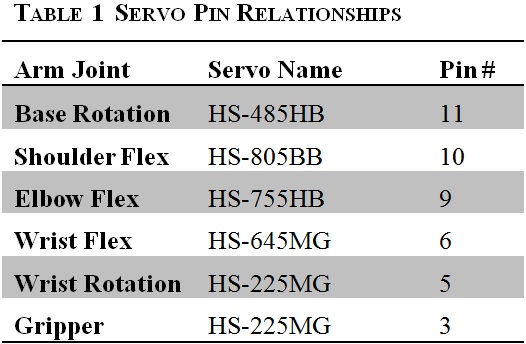
Arduino

UNO

Figure .1-1 System Block Diagram

## Setup Instructions

The microcontrollers of each subsystem should be mounted inside of the front panel of the rover. Then the serial connections should be made between the software and sensors, and the software and arm. The software will also need to be connected to the rovers display device using UDP. The servos will also need to be connected to the servos and arm microcontroller. The microcontroller used here is the BotBoarduino. Each servo has a three holed female connector. The yellow wire will plug into the innermost pin on the BotBoarduino. Table 1 displays the pin numbers that each servo connects to. The power supply then has to supply power to each subsystem. The power supply will have tags on the output to indicate which wire goes to where. There will be three wires from the power supply module, and an input wire to the Arduino DUE from the digital multimeter.



## Operating Instructions

Once the Autonomous Arm System is setup and installed, the operating instructions become very simple. The rover is not allowed to be in motion while the arm is away from the home position, or while a rock is in the gripper. The power supplies for the arm and rover communication device should be switched on before doing anything else. If the arm turns on and the forearm is at a 90 degree angle with the upper arm, but the beacon message says it is at home, send the save rock command. This will tell the arm to perform the motions to save a rock, even though there should not be a rock in the gripper. After the arm finishes the save rock action, it will return to its true home position. Now the rover motors system can be activated and drive around to search for rocks.   
 There are some stipulations to the environment of the system for the arm to function properly. The color of the items to be retrieved must be yellow, red, or blue. The ground on which the rover is navigating must be a color that contrasts to the retrieval items. Also, the items to be retrieved must be smaller than 3-inch cubes, and larger than 1-inch cubes. Items outside of these stipulations may be attempted to be retrieved, but there are no guarantees on the success rate of the arms functionality during or after any one of these attempts. If the arm fails, turn off the power to all systems and restart from the beginning of these instructions.  
 As for the process of picking up a rock, bring the rover to a halt first. The rover must be motionless before attempting to pick up any items. After coming to a halt, send the command to pick up a rock. The user will then be prompted to select the color of which rock to be picked up. Select the color by sending “Y”, “R”, or “B” back to the arm. After selecting the color, the arm will locate the rock and decide if it is within reach or not. If the rock is not within reach, the arm will send a beacon message saying “OUT OF REACH”. Wait until the arm sends the “HOME” beacon and move the rover closer to the rock and start again. If the rock is determined to be within reach, the arm will proceed to pick up a rock. This time the user will be prompted to decide whether the correct rock was retrieved. If the correct rock was retrieved, send the command “S” to the arm to save the rock. If the incorrect rock was retrieved, send the command “N” to the arm to drop the rock. After saving or dropping the rock, the arm will eventually send a beacon message saying that the arm is at home. Once this happens, the rover is free to send another pick up rock command or begin searching again for more rocks.   
 Below is chronological list of the operating instructions described in this section.

1. Turn on power to arm system and rover communication
2. Check the orientation of upper arm and forearm, if at a right angle send save rock command (“S”)
3. Maneuver the rover such that a rock is located in front of the rover
4. Bring the rover to a complete halt
5. Send the pick up rock command
6. Select the color of the rock to be picked up

7.1 If the rock is out of reach, return to Step 3 after the arm returns home

7.2 If the rock is in range, wait for arm to pick up rock and ask if the correct rock was retrieved

1. Send “S” to save the rock, or send “N” to discard the rock in the gripper
2. Wait until the arm returns home and return to step 3 or 5

# Subsystem Chapter

## Software

### Technical Overview of Subsystem

After the user stops the rover and sends the command to pick up the rock, the message will be sent to the Arduino microcontroller through an Ethernet connection. The microcontroller will then receive the coordinate information from the camera sensor and send the coordinate to the servo arm. After the servo turns toward the rock by the coordinate position it will request for the distance. The laser sensor will then retrieve the distance information and send it through the main microcontroller. The servo will determine whether the given distance is within the reaching range of arm kit, and if the distance is too far, the microcontroller will ask the user to move the rover closer, if the rock is within range, it will pick up the rock and finish the task. Each situation will update to the PC to let the user know the working status.

## Communication with Robotic Arm

The Robotic Arm (RA) will use UDP to transmit and receive messages between the RA and the Rover. We will not be using the RobotOpen protocol as previously stated in order to simplify the communication process.

Setup parameter should be stored in a setup file that are read into the program during the program startup, i.e. in setup(). This will allow the team to change IP addresses and port in the future without needing to make program changes.

To start with we will assume defaults as:

* Rover IP: 10.0.0.55 (this could change in the future)
* Robotic Arm IP: 10.0.0.5 (this could change in the future)
* We both will monitor Port: 8911 (this could change in the future)

## Connecting to Ethernet

* Use Ethernet.begin(mac, ip); Ethernet Library – this is the mac and IP address of the Ethernet shield.
* Use Udp.begin(localPort); to specify port - EthernetUDP

### Block Diagram

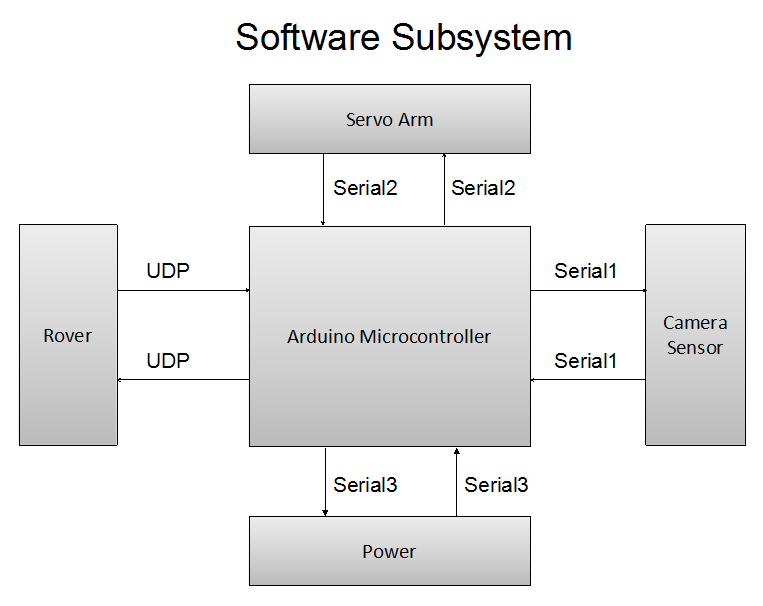


Figure 2.1-1 Sensors Block Diagram

### Flow Chart

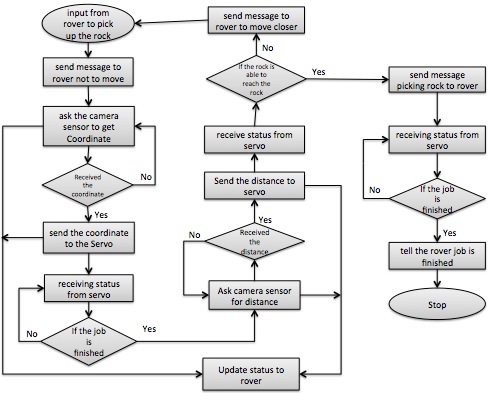


Figure 2.1-2 Software Flow Chart

### Manufacturing Tests and Equipment List

|  |  |
| --- | --- |
| Item # | Equipment List |
| 1 | Arduino Due Microcontroller |
| 2 | Arduino Ethernet Shield |

Table 4.1-1 Equipment List

### Parts Costs and Power Requirements

|  |  |  |
| --- | --- | --- |
| Parts | Item #’s (Equipment List) | Cost |
| Arduino Due board | 1 | $59.98 |
| Arduino Ethernet Shield | 2 | $34.98 |
| TOTAL |  | $94.96 |

Table 4.1-2 Parts Costs

Power input: 7V 0.8A

## Sensors

### Technical Overview of Subsystem

Image sensors are cheap and flexible choices for object detection. With the right algorithm, an image sensor can sense or detect practically anything. However there are two drawbacks with image sensors: 1) they output lots of data, dozens of megabytes per second, and 2) processing this amount of data can overwhelm many processors. And even if the processor can keep up with the data, much of its processing power won't be available for other tasks.

Therefore a dedicated processor will be assigned to complete the computationally intensive task of image processing. A product called Pixy addresses these problems perfectly. Pixy processes images from the image sensor and only sends the useful information (e.g. blue ball detected at x=23, y=54) to the microcontroller, at the frame rate of 50 Hz. The on-chip microprocessor performs all the image processing tasks. The information is able to be transmitted through SPI interface. So microprocessors like Arduino can communicate easily with Pixy without sacrificing lots of computing power.

Pixy outperforms many other vision sensors by utilizing a hue-based color filtering algorithm for object detection. Pixy calculates the hue and saturation of each RGB pixel from the image sensor and uses these as the primary filtering parameters. The hue of an object remains largely unchanged with changes in lighting and exposure. Changes in lighting and exposure can have a frustrating effect on color filtering algorithms. Pixy’s filtering algorithm is robust when it comes to lighting and exposure changes and significantly better than previous versions or other vision sensors.

The raw data is then interpreted in Arduino UNO and packed into a string and then transmitted through UART serial to the main microcontroller Arduino DUE. Since only the color and coordinate values are the only data going to be sent out and all the heavy processing is handled by the on-chip processor, the main controller has a lot of computing power for other tasks.

### Block Diagram

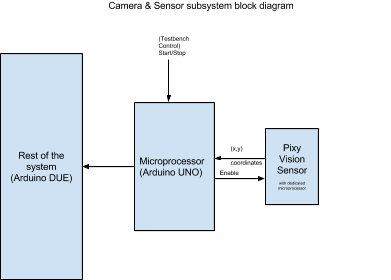


Figure 2.2-1 Sensors Block Diagram

### Flow Chart

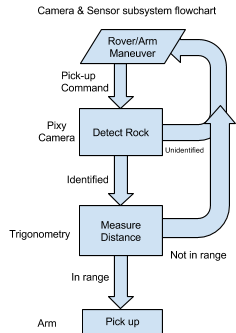


Figure 2.2-2 Sensors Flow Chart

### Schematics

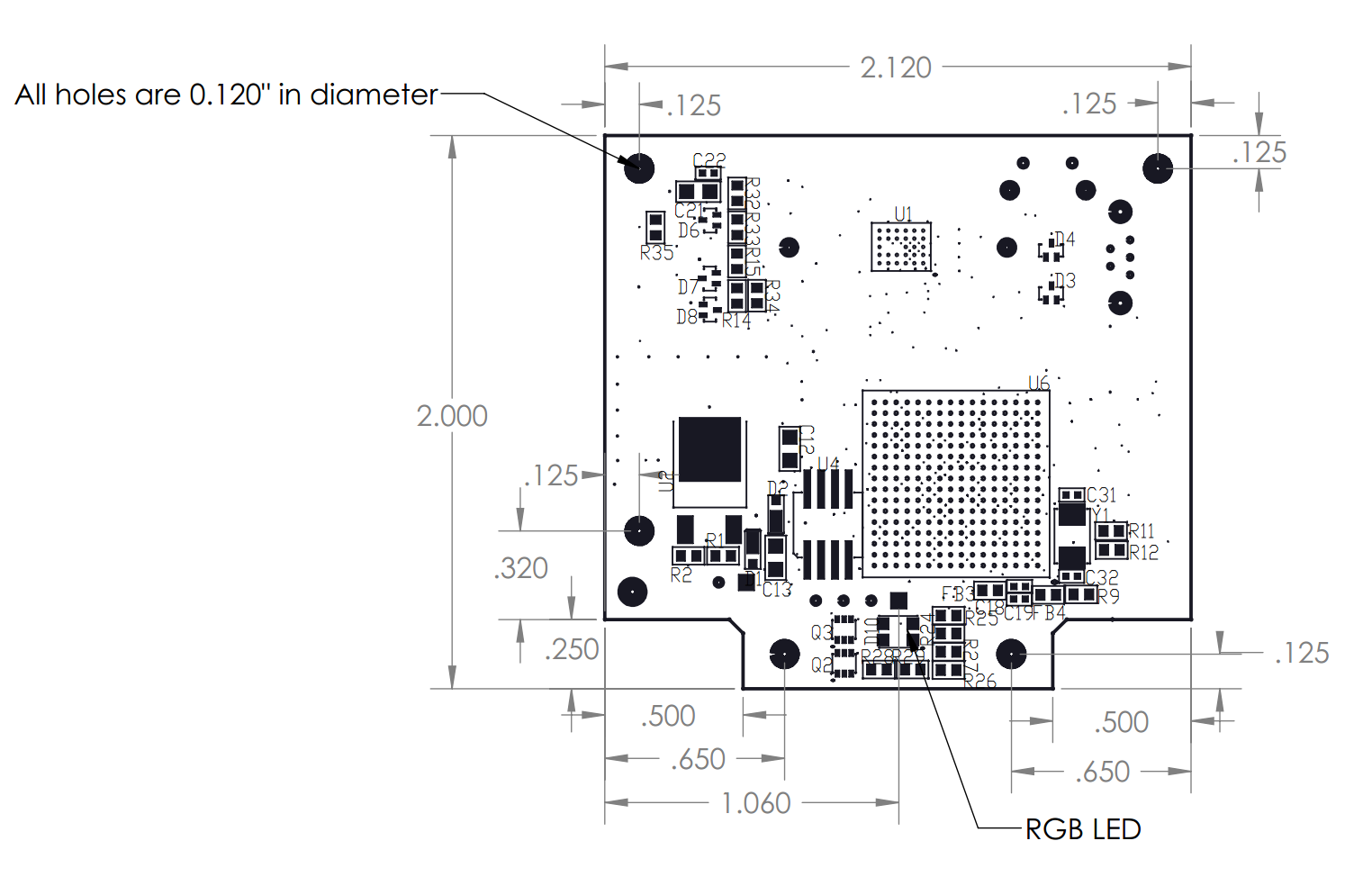


Figure 2.2-3 PIXY Camera Schematic

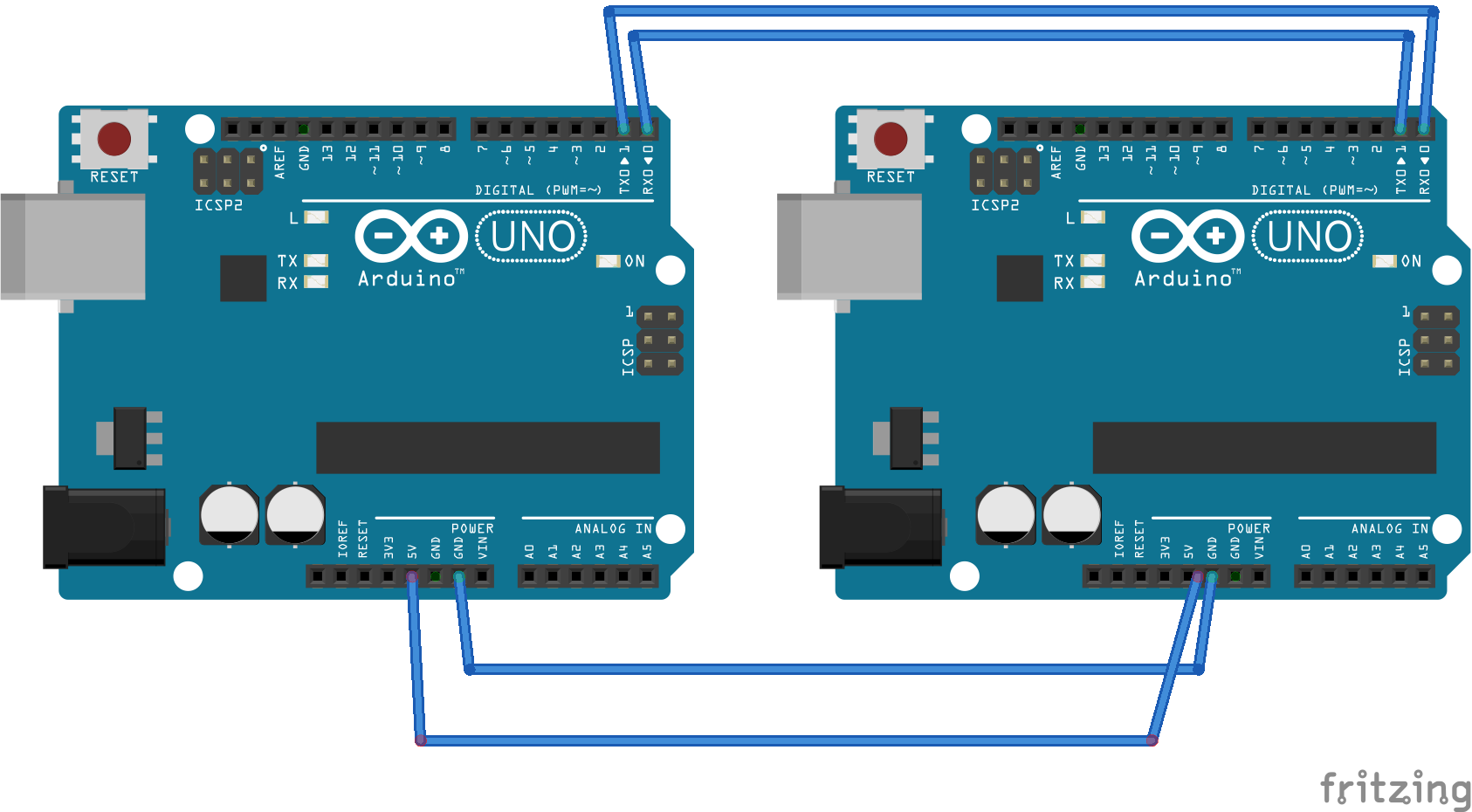


Figure 2.2-4 Serial Connection Schematic

### Manufacturing Tests and Equipment List

|  |  |
| --- | --- |
| Item # | Equipment List |
| 1 | Arduino UNO |
| 2 | Pixy Camera |

Table 2.2-1 Equipment List

Camera & Sensor Subsystem Instruction:

1. Take lens cap off.
2. Point the camera at a white object (AWB).
3. Connect Arduino UNO with Arduino DUE by UART serial.
4. Connect Pixy with Arduino UNO using the cable.
5. Hold down the button on top of Pixy. After about 1 second, the LED will turn on, first white, then red. When it turns red, release the button.
6. After releasing the button, Pixy will enter “learning” mode, where the LED color is the color of the center pixels of Pixy's image frame.
7. Use the LED color as feedback to determine if the object is in the center of Pixy's image frame. When you are satisfied that the LED color matches your object color press and release the button.
8. If Pixy determines that the quality (hue) of the object is "good enough" (has enough color saturation), the LED will flash and Pixy has remembered that particular color. If the LED directly goes off, it means the quality is not good enough for detection, then you need to try it again in a better light condition or a more fluorescent object.
9. If you want Pixy learn another color, redo step 5 - 8.

### Parts Costs and Power Requirements

|  |  |
| --- | --- |
| Parts | Cost |
| PIXY (CMUcam5) vision sensor (5V) | $69.00 |
| TOTAL | $69.00 |

Table 2.2-2 Parts Costs

Power consumption: 140 mA typical

Power input: USB input (5V) or unregulated input (6V to 10V)

## Servos and Arm

### Technical Overview of Subsystem

The Servos and Arm Subsystem functions off of the BotBoarduino microcontroller that sends signals to the arms six servos. The program encoded to the BotBoarduino is the real work horse of this subsystem. The actual code can be found in the appendix. A brief description of the program is provided here.   
 The program initializes a setup routine that puts the arm into the home position, which is used as a reference point many times in the functions of the arm. After setup, the program runs a loop that is constantly looking for messages. If the BotBoarduino receives an “H”, this sets a flag that runs a function to put the arm into the home position. If a “C” is received, another flag is set and the BotBoarduino then expects to receive coordinates that help the arm point its gripper directly at the rock to be picked up. It then goes directly to another function that calculates the exact location of the rock, repositions itself to grip the rock, then carries the rock back to the home position. A message is then sent to the rover asking if the correct rock was retrieved. The program goes back to the loop, waiting for a message to be received signifying whether or not the correct rock was retrieved. Based on the reply, the arm sets a flag to either discard the rock or save the rock by placing it into a basket located on top of the rover.  
 A much more detailed description of these functions can be found in the evidence section of the System Report.

### Block Diagram

Figure 2.3-1 Servos and Arm Block Diagram

The microcontroller used in this subsystem is the BotBoarduino. It uses one serial connection to communicate with the software subsystem. More detail on the messages will be discussed in section 4.3.2. The BotBoarduino contains six PWM generators, which are used to control the overall orientation of the arm. The orientation of the arm will be determined based on the calculations performed in the microcontroller.

### Flow Chart

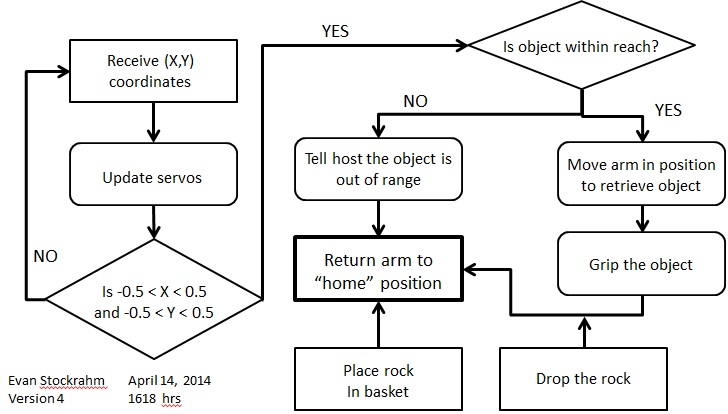


Figure 2.3-2 Servos and Arm Flow Chart

Notice the blocks with square corners. These blocks signify starting points of the flow chart based on which message is received. When “C” is received, the arm goes through the flow chart starting at the top left squared block. When “S” is received to save the rock, the arm starts at the block labeled, “Place rock in basket”. When “N” is received to discard the rock, the arm starts at the block labeled “Drop the rock”.

|  |  |
| --- | --- |
| Message (In/Out) | Message Description |
| “H” (In) | Go to the home position |
| “C” (In) | Receive coordinates and locate the rock |
| “S” (In) | Save the rock that has been retrieved |
| “N” (In) | Do not keep the rock in the gripper |
| “?” (Out) | Asks the rover if the correct rock was retrieved |
| Beacon Message (Out) | 1 of 7 messages that flash on the rover’s screen notifying the rover of what the arm is currently doing |

Table 2.3-1 Message Descriptions

### Manufacturing tests and Equipment list

|  |  |
| --- | --- |
| Item # | Equipment List |
| 1 | AL5D Robotic Arm Hardware Kit |
| 2 | BotBoarduino  Table 2.3-2 Equipment List |
| 3 | HS-225MG (gripper servo) |
| 4 | HS-225MG (wrist rotate servo) |
| 5 | HS-645MG (wrist flex servo) |
| 6 | HS-755HB (elbow flex servo) |
| 7 | HS-805BB (shoulder flex servo) |
| 8 | HS-485HB (base rotate servo) |
| 9 | Wrist Rotate Upgrade (Medium Duty) |
| 10 | A Style Gripper |

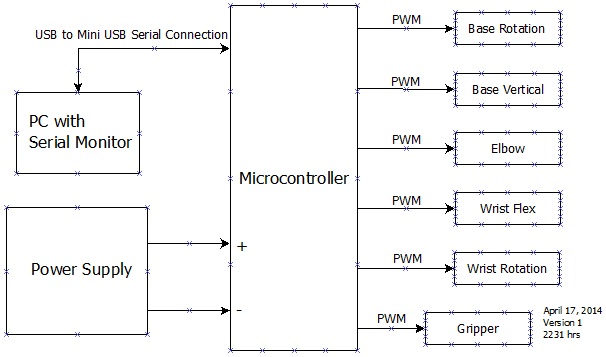


Figure 2.3-3 Manufacturing Test Block Diagram

The PC will be used to provide commands and messages to the subsystems microcontroller. It will also respond to any inquiries by the subsystem. The power supply must provide 4.8-6 Volts and be able to support at least 2 amps worth of current.

### Parts Costs and Power Requirements

|  |  |  |
| --- | --- | --- |
| Parts | Item #’s (Equipment List) | Cost |
| AL5D Robotic Arm Combo  Kit (BotBoarduino) | 1,2,3,5,6,7, 8 | $309.81 |
| Wrist Rotate Upgrade | 4 and 9 | $36.34 |
| A Style Gripper | 10 | $50.95 |
| TOTAL  Table 2.3-3 Parts Costs |  | $397.10 |

The Servos and Arm Subsystem requires 4.8-6 Volts with a load of at least 2 Amps.

## Power Supply

### Power Supply Overview

The power supply subsystem will have 3 parts to its system. First, the system will have a battery fuel gauge between the battery and the different switching power supply modules. This fuel gauge will show the remaining battery state of charge via the outputs. Normally these outputs are used to power LEDs to indicate the state of charge, but in our case we will use these outputs to digitals inputs in the main MCU and indicate the remaining charge to the user.

Second, there will be 3 power supply modules for the different parts of the product. The servos will be powered by one module with output at 6V and 2A max. The Arduino Due and the Pixy camera will share the same module with output at 7V and 1A max. The digital multimeter using MSP430F5529 will be powered by another module that is connected to the 7V module for the Pixy and Due with output of 5V and 22mA.

Third, the subsystem will have its own monitoring tool. The tool will use a microcontroller to evaluate the voltage and current going out of the power supply modules. Voltage will be measured directly by the analog input in the micron controller while the current will be measured by a 0.250 mOhm resistor and calculated by the measured voltage.

### Block Diagram

Figure 2.4-1 Power Supply Block Diagram

7V 0.15A

7V 0.8A

5V 22mA

7V 22mA

6V 2A

12.8V 3300mAh

### Schematic

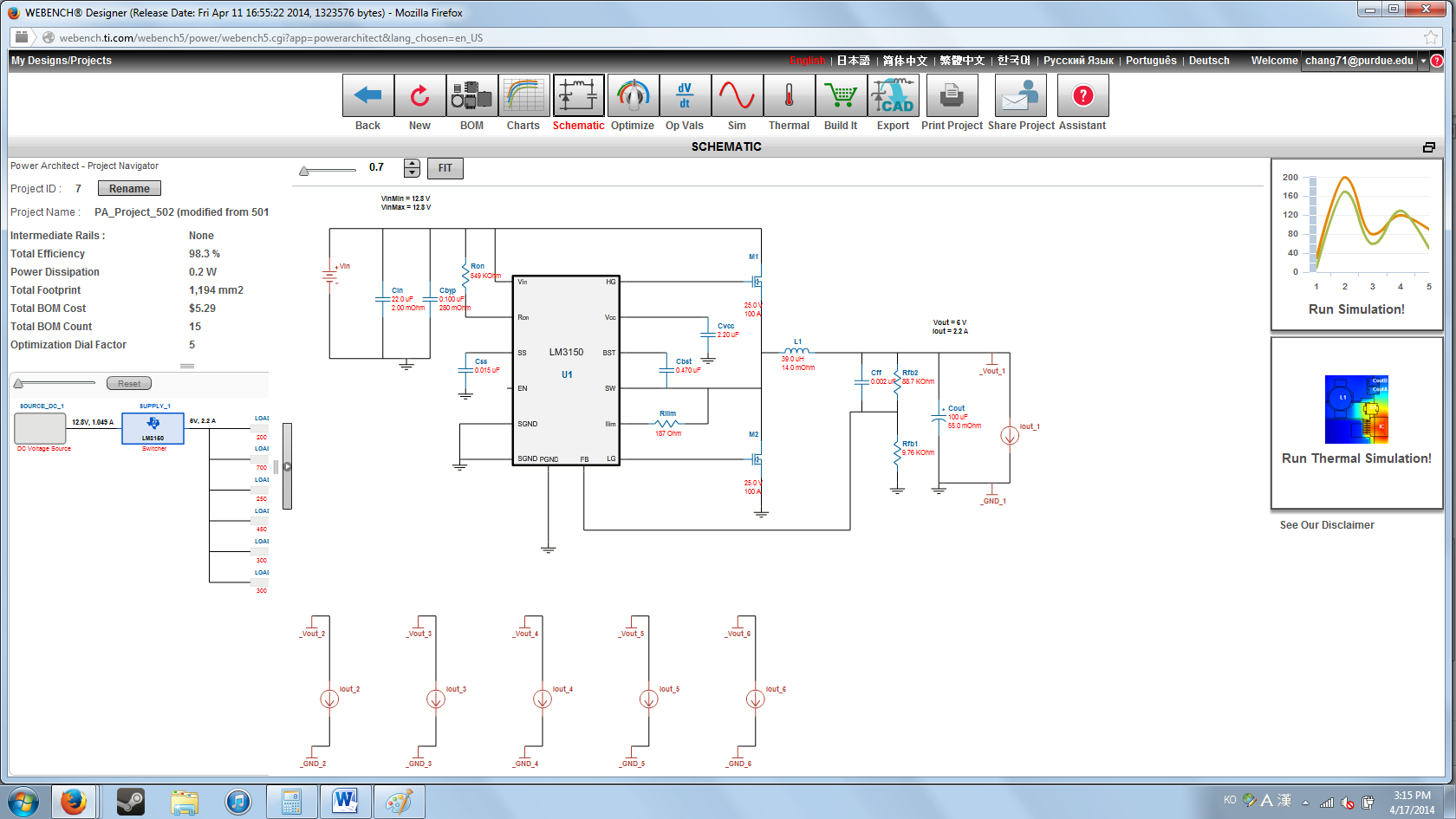


Figure 2.4-2 6V 2A output switching module

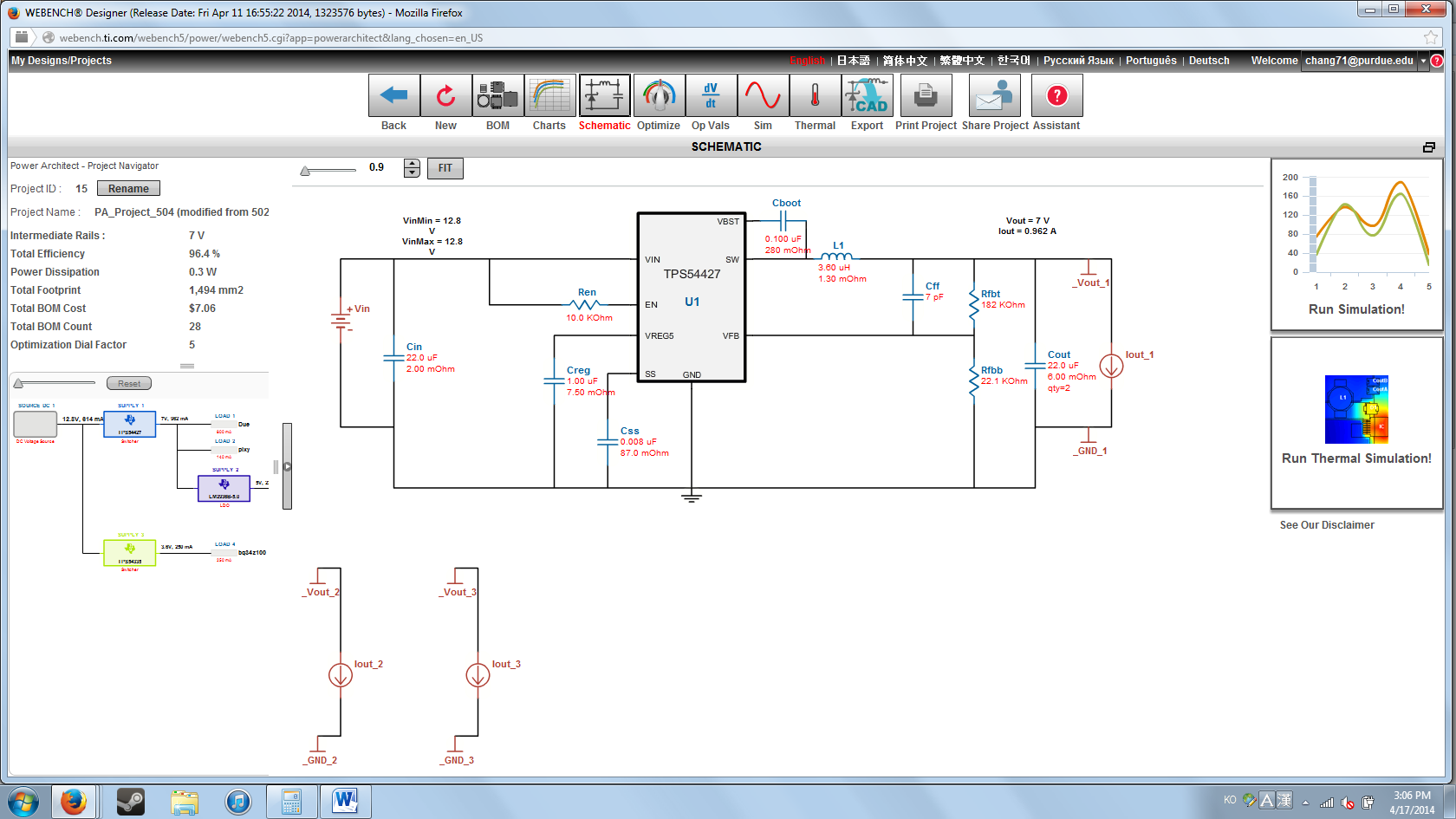


Figure 2.4-3 7V 1A output switching module

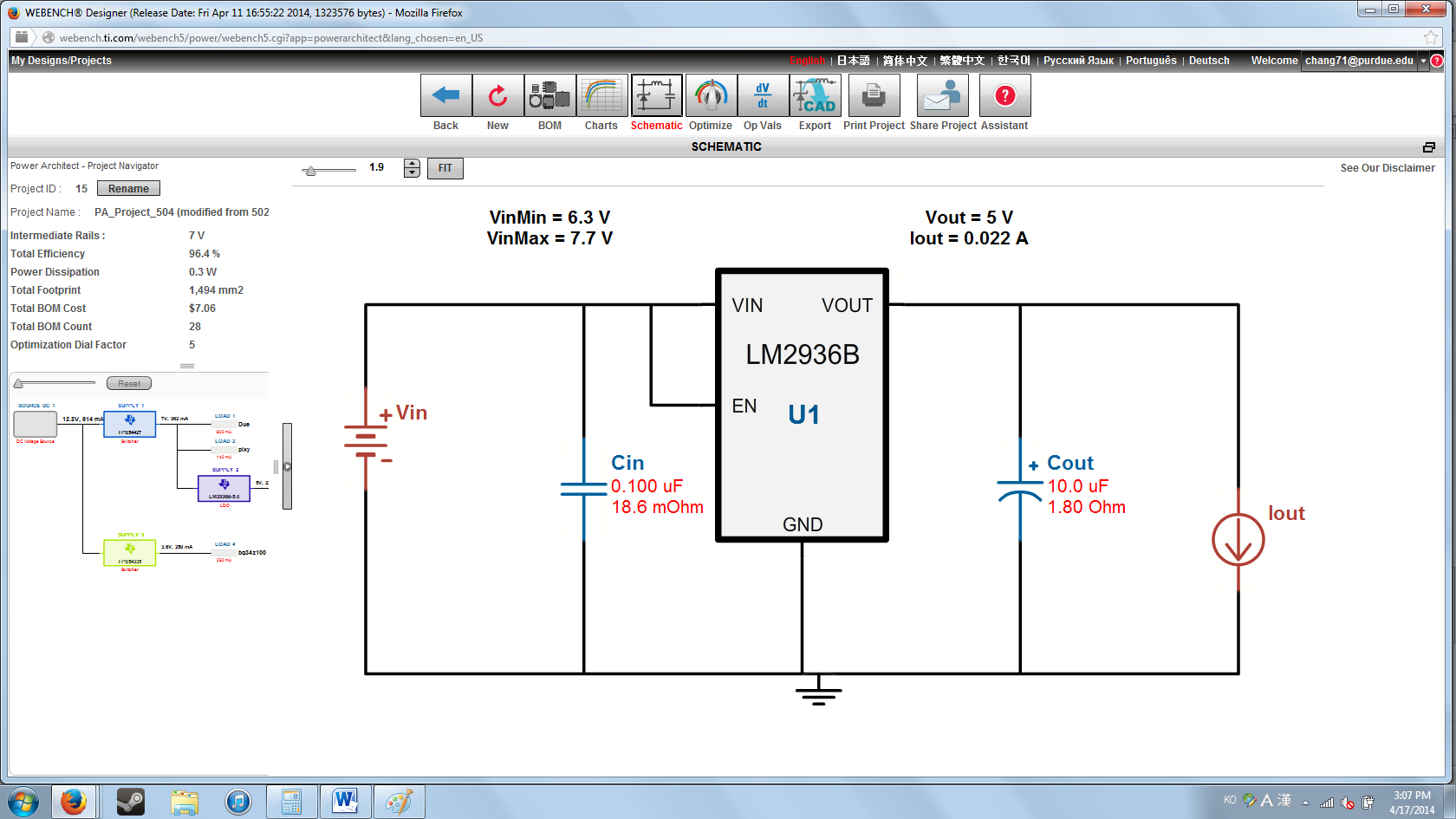


Figure 2.4-4 5V 22mA output switching module

### Manufacturing Test and Equipment List

|  |  |
| --- | --- |
| Item # | Equipment List |
| 1 | Computer w/ energia code editor |
| 2 | micro-USB cable  Table 2.4-1 Equipment List |

Connection Diagram:

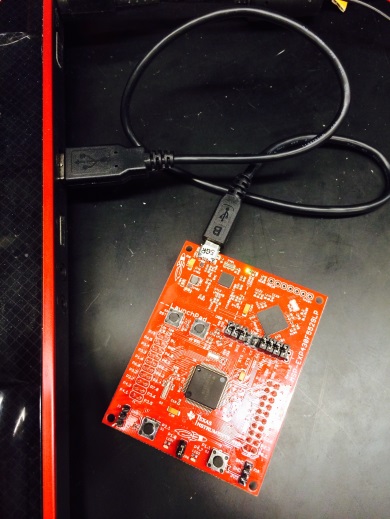


Figure 2.4-5 Connection Diagram

A computer with energia code editor(download for free from [www.energia.nu](http://www.energia.nu)) and micro-USB cable is needed for the manufacturing test. Connect the microcontroller to the computer via the micro-USB cable. Open energia code editor, go to Tools and select Serial Monitor. Since the computer powers the microncontroller, the serial monitor should be automatically updating the measured multimeter values. The multimeter will come connected to the output of each power supply. So, by just connecting the battery to the power supply modules you will be able to check the output voltage and output current in each module. The module outputs should be 6V for Servo, 7V for the Arduino Due and Pixy Camera and 12.8V for the Battery. With no load, the current should be 50mA or lower.

The battery fuel gauge will come with its own LED outputs. These outputs will also be connected to the inputs of the Arduino Due in the final product to be sent to the user. If the battery is fully charged, all four LEDs should light up with each LED turning off for 25% less remaining state of charge.

|  |  |
| --- | --- |
| Part | Cost |
| 6V switching power supply | $20.20 |
| 7V switching power supply | $3.24 |
| 5V linear power supply | $0.78 |
| MSP430F5529 | $12.99 |
| BQ34Z100EVM | $49.99 |
| LiFEPO4 12.8V 3300mAh Rechargeable Battery | $55.99 |
| Tenergy 12.8V (4-Cell) Intelligent 1.5A LiFePO4 Battery Pack Charger | $29.99 |
| TOTAL | $173.18 |

### Parts Costs and Power Requirements

Table 2.4-2 Power supply cost

Power input: 12.8V 3300mAh

###### Example Appendix

Servo and Arm Subsystem BotBoarduino Code

/\* The purpose of this program is to pick up a rock that has

been located by a camera. As long as a camera is mounted to the

wrist of the arm, and x,y coordinates are recieved, this program

will move an arm with 5 DOF and a gripper to the rock and pick

it up and place it in a basket behind the arm. \*/

#include <Servo.h>

#include <EEPROM.h>

#include <math.h>

Servo ShoulderRot;

Servo ShoulderFlex;

Servo ElbowFlex;

Servo WristFlex;

Servo WristRot;

Servo Gripper;

int sRot = 90; //angle at which the base is rotated

int sFlex = 135; //angle at which the shoulder is flexed

int eFlex = 155; //angle at which the elbow is flexed

int wrFlex = 130; //angle at which the wrist is flexed

int wrRot = 90; //angle at which the wrist is rotated

int grip = 150; //gripper opening variable

int theta = 65; // angle between ground and gripper opening

float const HT = 10; //height of the gripper opening at home position in inches

float const GLOFST = 4.5;

float const UA = 5.75; // length of upper arm

float const FA = 7.375; // length of forearm

float a; // the horizontal distance from wrist flex joint to rock

float f; // the horizontal distance from shoulder joint to wrist

float h; // hypotenuse of the shoulder-wrist triangle, used to find g

float g; // the vert distance from shoulder to wrist joint

float D; // the distance the shoulder needs to be from the wrist in order to pick up rock

int const HsROT = 90; // home pos angle for base rotation

int const HsFLEX = 135; // Home pos angle for shoulder

int const HeFLEX = 155; // Home pos angle for elbow

int const HwrFLEX = 130; // Home pos angle for wrist flex

int const HwrROT = 90; // Home pos angle for wrist rotate (perpindicular to ground)

int const HGRIP = 150; // Home pos opening for gripper (closed)

float const REACH = 13.5; // Maximum reach for the arm

int const HrtBtInterval = 2000; // beacon interval

int state; // used to notify user of what state the rover is in

int homeflag = 0; // flag for going to home position

int zeroing = 0; // zeroing flag

int getrock = 0;

int wrongrock = 0;

int saverock = 0;

int sRotNew; //angle at which the base needs to be rotated

int sFlexNew; //angle at which the shoulder needs to be flexed

int eFlexNew; //angle at which the elbow needs to be flexed

int wrFlexNew; //angle at which the wrist needs to be flexed

int wrRotNew; //angle at which the wrist needs to be rotated

int gripNew; //value at which the gripper needs to be opened/closed to

int nextHeartBeatms = 0;

struct coord

{

float x;

float y;

};

void setup()

{

Serial.begin(9600);

ShoulderRot.attach(11); // attaches the servos to their pins

ShoulderFlex.attach(10);

ElbowFlex.attach(9);

WristFlex.attach(6);

WristRot.attach(5);

Gripper.attach(3);

nextHeartBeatms = millis() + HrtBtInterval;

homepos();

}

void loop()

{

readMessage(); // calls function to read which state I need to be in

if (homeflag)

{

homepos(); // this function goes to home position

}

if (zeroing)

{

xyZero(); // need to figure out struct coord to read the xy-coordinates

//then relocating should be simple

}

if (getrock)

{

pickup(); // need to create a fxn to determine if rock is in range

// does the trig for and puts arm in position to grip the rock

// also sends the arm home after picking up the rock

}

if (wrongrock)

{

droprock(); // opens the gripper to drop the rock then goes home again

}

if (saverock)

{

placerock(); // places rock 'behind' arm into basket and goes home again

}

beacon();

delay(2000);

}

void beacon()

{

if (nextHeartBeatms <= millis())

{

Serial.print("Time: ");

Serial.println(millis());

//Serial.print("\t");

//Serial.println(state);

if (state == 72)

{

Serial.println("Home"); // write which mode I'm in to Rover

}

else if (state == 67)

{

Serial.println("Locating");

}

else if (state == 71)

{

Serial.println("Picking up rock");

}

else if (state == 83)

{

Serial.println("Saving rock");

}

else if (state == 78)

{

Serial.println("Dropping rock");

}

else if (state == 1)

{

Serial.println("Moving Home");

}

else if (state == 2)

{

Serial.println("Out of Reach");

homeflag = 1;

}

else if (state == 3)

{

Serial.println("Correct Rock?");

}

nextHeartBeatms = millis() + HrtBtInterval;

}

}

void homepos()

{

sRotNew = HsROT;

sFlexNew = HsFLEX;

eFlexNew = HeFLEX;

wrFlexNew = HwrFLEX;

wrRotNew = HwrROT;

gripNew = HGRIP;

state = 1;

updateServos();

state = 72;

homeflag = 0;

}

// Used to zero in on the (x,y) coordinate for a constant update

void xyZero()

{

sRotNew = 108;

wrFlexNew = 95;

updateServos();

zeroing = 0;

getrock = 1;

}

// Used to find new angles to pick up the rock

// the goal of this function is to reassign angles for each joint

// such that the gripper is hovering directly over the rock

void pickup()

{

Gripper.write(0); // open the gripper

gripNew = 0;

grip = gripNew;

//wrFlex = 90; // resetting the wrFlex to simulate the camera being xyZero'd

theta = 180 - HeFLEX + wrFlex - 90; // angle between ground and gripper opening

Serial.print("theta = ");

Serial.println(theta);

a = 10 \* tan(theta\*PI/180); // the horizontal distance from wrist flex joint to rock

Serial.print("a = ");

Serial.println(a);

f = FA \* sin((180-HeFLEX)\*PI/180); // the horizontal distance from shoulder joint to wrist

Serial.print("f = ");

Serial.println(f);

h = sqrt(UA\*UA + FA\*FA - 2\*UA\*FA\*cos((180-HeFLEX)\*PI/180)); // hypotenuse of the shoulder-wrist triangle, used to find g

Serial.print("h = ");

Serial.println(h);

g = sqrt(h\*h -f\*f); // the vert distance from shoulder to wrist joint

Serial.print("g = ");

Serial.println(g);

D = sqrt((a+f)\*(a+f) + (g+HT-GLOFST)\*(g+HT-GLOFST)); // the distance the shoulder needs to be from the wrist in order to pick up rock

Serial.print("D = ");

Serial.println(D);

if (D > REACH)

{

state = 2;

Serial.write("Out of reach");

}

else if (D <= REACH)

{

eFlexNew = acos((D\*D - UA\*UA - FA\*FA)/(-2\*FA\*UA))\*180/PI;

Serial.print("eFlexNew = ");

Serial.println(eFlexNew);

sFlexNew = HsFLEX - (180 - atan((a+f)/(g + HT - GLOFST))\*180/PI - asin(FA\*sin(eFlexNew\*PI/180)/D)\*180/PI);

Serial.print("sFlexNew = ");

Serial.println(sFlexNew);

wrFlexNew = 225 - (sFlexNew + 180 - eFlexNew);

Serial.print("wrFlexNew = ");

Serial.println(wrFlexNew);

state = 71;

updateServos();

gripNew = 180; // close the gripper to grasp rock, arm should be in position at this point

wrRotNew = 0;

updateServos();

homepos();

state = 3;

}

getrock = 0;

}

void droprock()

{

gripNew = 0; // open the gripper all the way to drop the rock

wrRotNew = 0;

updateServos();

homepos();

wrongrock = 0;

}

void placerock()

{

sRotNew = 90;

sFlexNew = 150;

eFlexNew = 0;

wrFlexNew = 170;

wrRotNew = 0;

gripNew = 0;

updateServos();

homepos();

saverock = 0;

}

/\*

void readCoord()

{

if (Serial.available() > 0)

{

struct coord

{

float x;

float y;

}

coord = Serial.read();

x = coord.x;

y = coord.y;

Serial.println(x);

Serial.println(y);

}

}

\*/

void readMessage()

{

if(Serial.available() > 0)

{

state = Serial.read();

Serial.println(state);

if (state == 72) // "H" = 72 in ASCII

{

homeflag = 1;

}

else if (state == 67) // "C" = 67 in ASCII

{

zeroing = 1;

}

else if (state == 71) // "G" = 71 in ASCII

{

getrock = 1;

}

else if (state == 83) // "S" = 83 in ASCII

{

saverock = 1;

}

else if (state == 78) // "N" = 78 in ASCII

{

wrongrock = 1;

}

}

/\* Serial.print(homeflag);

Serial.print(zeroing);

Serial.print(getrock);

Serial.print(saverock);

Serial.println(wrongrock);

\*/}

// Used to update all servos to global variables

void updateServos()

{

if(sRotNew > sRot) //rotate base to the right

{

for(sRot = sRot; sRot < sRotNew; sRot += 1) // goes from 0 degrees to 180 degrees

{ // in steps of 1 degree

ShoulderRot.write(sRot); // tell servo to go to position in variable 'pos'

beacon();

delay(15); // waits 15ms for the servo to reach the position

}

}

else if(sRotNew < sRot) //rotate base to the left

{

for(sRot = sRot; sRot > sRotNew; sRot-=1) // goes from 180 degrees to 0 degrees

{

beacon();

ShoulderRot.write(sRot); // tell servo to go to position in variable 'pos'

delay(15); // waits 15ms for the servo to reach the position

}

}

else if(sRot == sRotNew)

{

}

if(sFlexNew > sFlex) //rotate base to the right

{

for(sFlex = sFlex; sFlex < sFlexNew; sFlex += 1) // goes from 0 degrees to 180 degrees

{ // in steps of 1 degree

beacon();

ShoulderFlex.write(sFlex); // tell servo to go to position in variable 'pos'

delay(15); // waits 15ms for the servo to reach the position

}

}

else if(sFlexNew < sFlex) //rotate base to the left

{

for(sFlex = sFlex; sFlex > sFlexNew; sFlex-=1) // goes from 180 degrees to 0 degrees

{

beacon();

ShoulderFlex.write(sFlex); // tell servo to go to position in variable 'pos'

delay(15); // waits 15ms for the servo to reach the position

}

}

else if(sFlex == sFlexNew)

{

}

if(eFlexNew > eFlex) //rotate base to the right

{

for(eFlex = eFlex; eFlex < eFlexNew; eFlex += 1) // goes from 0 degrees to 180 degrees

{ // in steps of 1 degree

ElbowFlex.write(eFlex); // tell servo to go to position in variable 'pos'

beacon();

delay(15); // waits 15ms for the servo to reach the position

}

}

else if(eFlexNew < eFlex) //rotate base to the left

{

for(eFlex = eFlex; eFlex > eFlexNew; eFlex-=1) // goes from 180 degrees to 0 degrees

{

ElbowFlex.write(eFlex); // tell servo to go to position in variable 'pos'

beacon();

delay(15); // waits 15ms for the servo to reach the position

}

}

else if(eFlex == eFlexNew)

{

}

if(wrFlexNew > wrFlex) //rotate base to the right

{

for(wrFlex = wrFlex; wrFlex < wrFlexNew; wrFlex += 1) // goes from 0 degrees to 180 degrees

{ // in steps of 1 degree

WristFlex.write(wrFlex); // tell servo to go to position in variable 'pos'

beacon();

delay(15); // waits 15ms for the servo to reach the position

}

}

else if(wrFlexNew < wrFlex) //rotate base to the left

{

for(wrFlex = wrFlex; wrFlex > wrFlexNew; wrFlex-=1) // goes from 180 degrees to 0 degrees

{

WristFlex.write(wrFlex); // tell servo to go to position in variable 'pos'

beacon();

delay(15); // waits 15ms for the servo to reach the position

}

}

else if(wrFlex == wrFlexNew)

{

}

if(wrRotNew > wrRot) //rotate base to the right

{

for(wrRot = wrRot; wrRot < wrRotNew; wrRot += 1) // goes from 0 degrees to 180 degrees

{ // in steps of 1 degree

WristRot.write(wrRot); // tell servo to go to position in variable 'pos'

beacon();

delay(15); // waits 15ms for the servo to reach the position

}

}

else if(wrRotNew < wrRot) //rotate base to the left

{

for(wrRot = wrRot; wrRot > wrRotNew; wrRot-=1) // goes from 180 degrees to 0 degrees

{

WristRot.write(wrRot); // tell servo to go to position in variable 'pos'

beacon();

delay(15); // waits 15ms for the servo to reach the position

}

}

else if(wrRot == wrRotNew)

{

}

if(gripNew > grip) //rotate base to the right

{

for(grip = grip; grip < gripNew; grip += 1) // goes from 0 degrees to 180 degrees

{ // in steps of 1 degree

Gripper.write(grip); // tell servo to go to position in variable 'pos'

beacon();

delay(15); // waits 15ms for the servo to reach the position

}

}

else if(gripNew < grip) //rotate base to the left

{

for(grip = grip; grip > gripNew; grip-=1) // goes from 180 degrees to 0 degrees

{

Gripper.write(grip); // tell servo to go to position in variable 'pos'

beacon();

delay(15); // waits 15ms for the servo to reach the position

}

}

else if(grip == gripNew)

{

}

Serial.println("Servos Updated");

Serial.print("sRot = ");

Serial.println(sRot);

Serial.print("sFlex = ");

Serial.println(sFlex);

Serial.print("eFlex = ");

Serial.println(eFlex);

Serial.print("wrFlex = ");

Serial.println(wrFlex);

Serial.print("wrRot = ");

Serial.println(wrRot);

Serial.print("grip = ");

Serial.println(grip);

}