Bluetooth Controllable Blimp

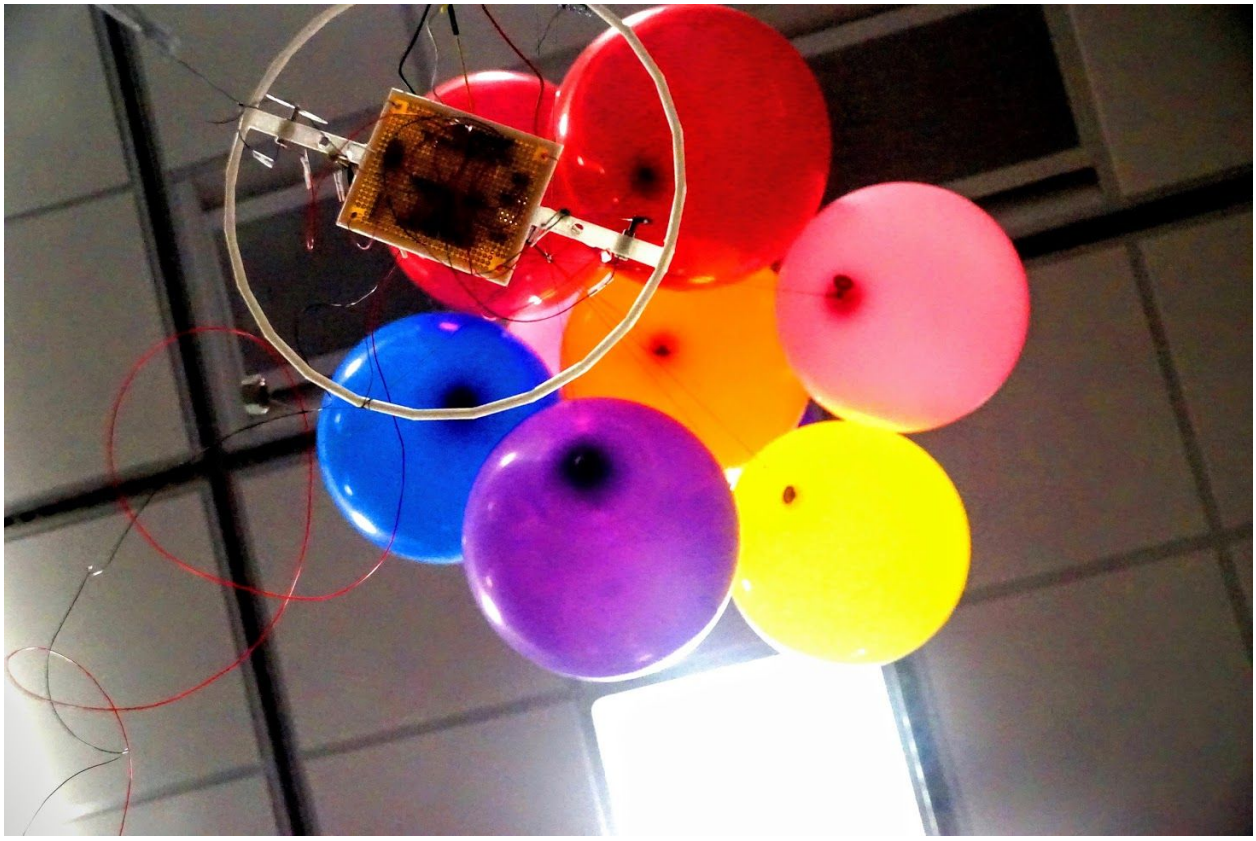
Soham Waychal
Vedanth Swain

under the direction of
Professor Lippold Haken
Chuanzheng 'Chad' Li
Aaron Smith

May 8, 2016

Abstract

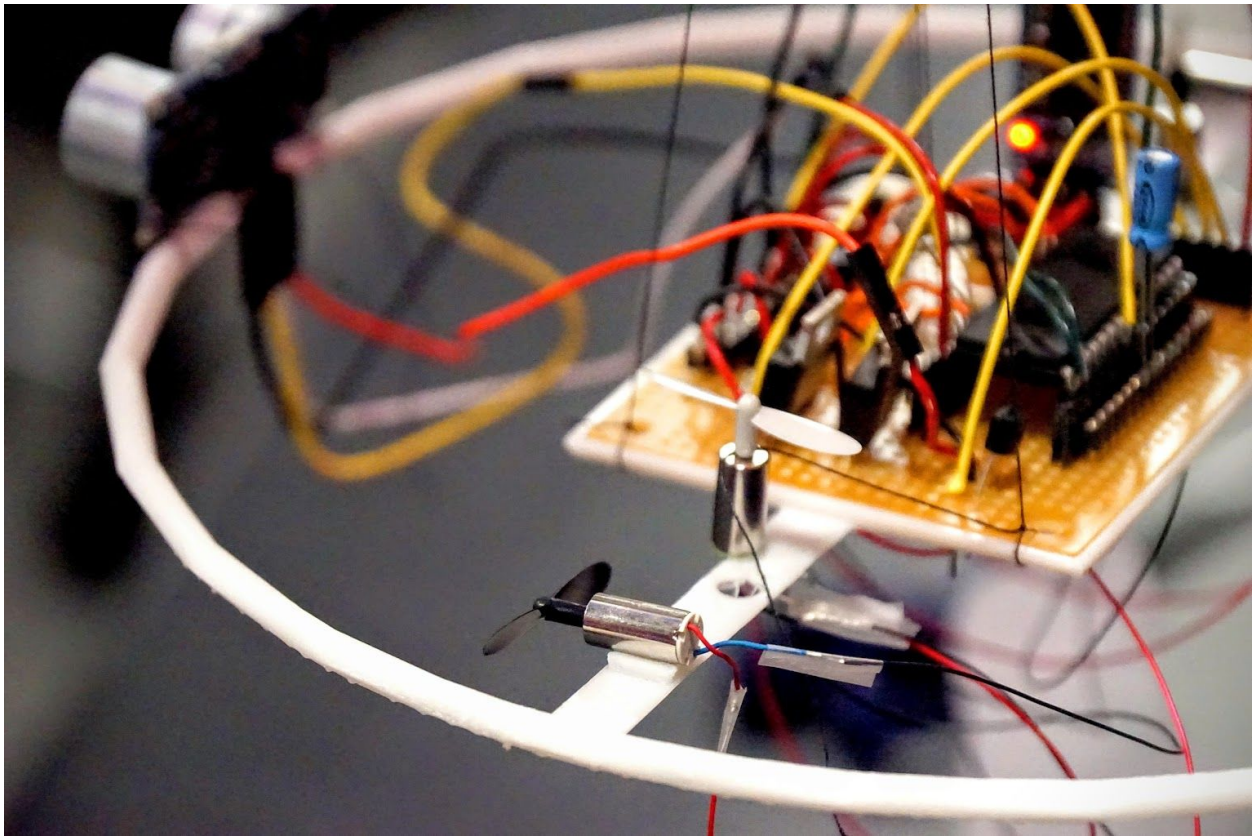
The original plan for this project was to build a quadcopter with a camera that would follow a selected target. After talking to Professor Haken, we changed the quadcopter to a blimp but retained the initial idea of using computer vision to control the blimp to follow a target. Halfway through the semester, after working on the camera for a while and frying it, we realized that we would not be able to finish the project in time and had to decide between building and controlling the blimp or working on the camera and getting some form of tracking going. We choose to build the blimp, mostly because we had fried our only camera, and control it using bluetooth and an android device. The final project that we demoed was a blimp that was 3D printed, had balloons tied to it to keep it afloat, had two motors for right, left, and forward control, and two motors for pushing the blimp up.



Introduction

The idea for this project originated freshmen year where we wanted to build a sort of pet robot that followed you around. The obvious choice for this was a quadcopter since they can go across any terrain and are very responsive machines. The choice of how they would follow included GPS location, a proximity sensor, and by using computer vision. We decided on computer vision as this could be extended in the future to incorporate many other things and could be used to make the quadcopter avoid obstacles and analyze the surroundings.

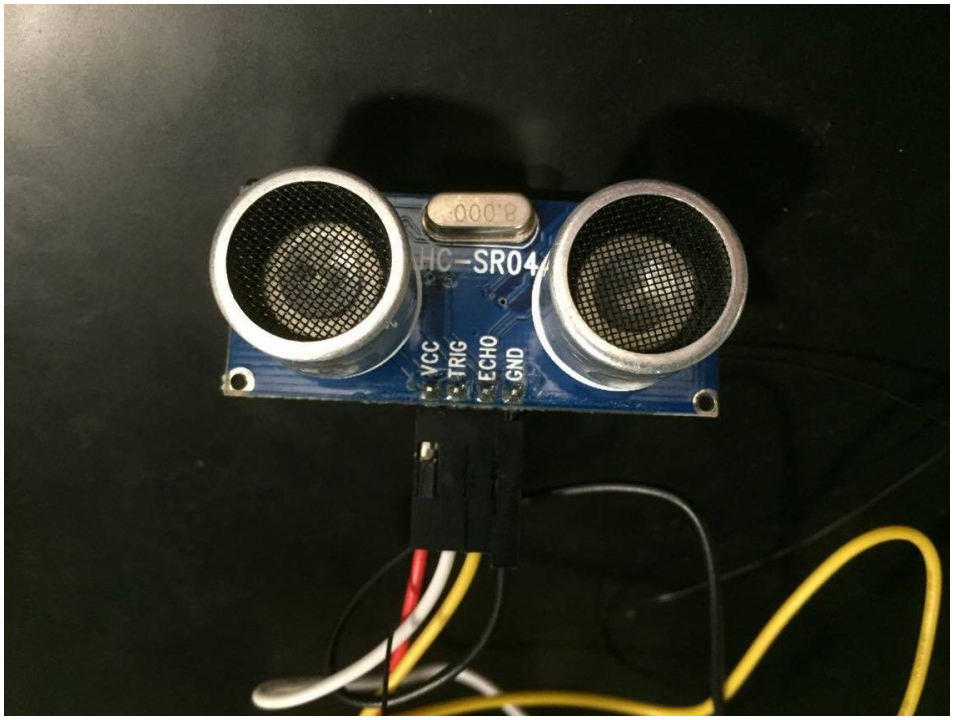
The project, as mentioned in the abstract, turned into a blimp that could be controlled via an android device and a distance sensor to keep the blimp from hitting obstacles. We used a HC-06 bluetooth module for communication between the android device and the microcontroller, a HC-SR04 distance sensor to detect if objects are within a certain range of the blimp, Estes 4617 motors to control the direction of the blimp, several transistors for the drive circuit for the motors, two 3.4 V LiPo batteries in series for a 7.4 V output, and an OV7670 camera for image capture. On top of that, we used party balloons and helium to make the blimp hover and 3D printed the frame of the blimp.



Distance sensor

The HC-SR04 ultrasonic sensor uses sonar to determine the distance of an object. It offers great range detection with good accuracy and stable readings. From 2cm to 400 cm. It operation is not affected by sunlight.

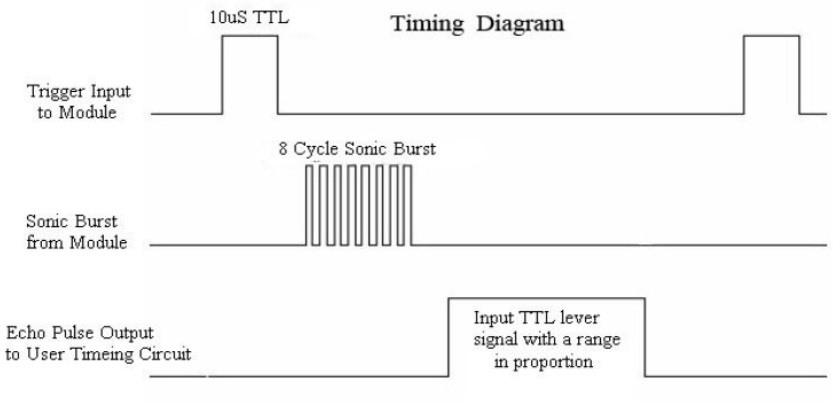
It has 4 pins: VCC = +5VDC
Trig = Trigger input of Sensor
Echo = Echo output of Sensor
GND = GND



Specification: Working Voltage - 5V
Working Current - 15mA
Working Frequency - 40 Hz

There are two key component to operating the HC-SR04 ('Trig' & 'Echo'). We need to supply a 10uS pulse to the trigger input, in order for it to start ranging. After that the module will send out 8 cycles bursts of ultrasound at 40KHz and then wait for the reflected burst. When the sensor detects ultrasound from the receiver, it will set the 'Echo' pin to high (5V) and delay for a period which is proportional to the distance. The equation for calculating distance is:

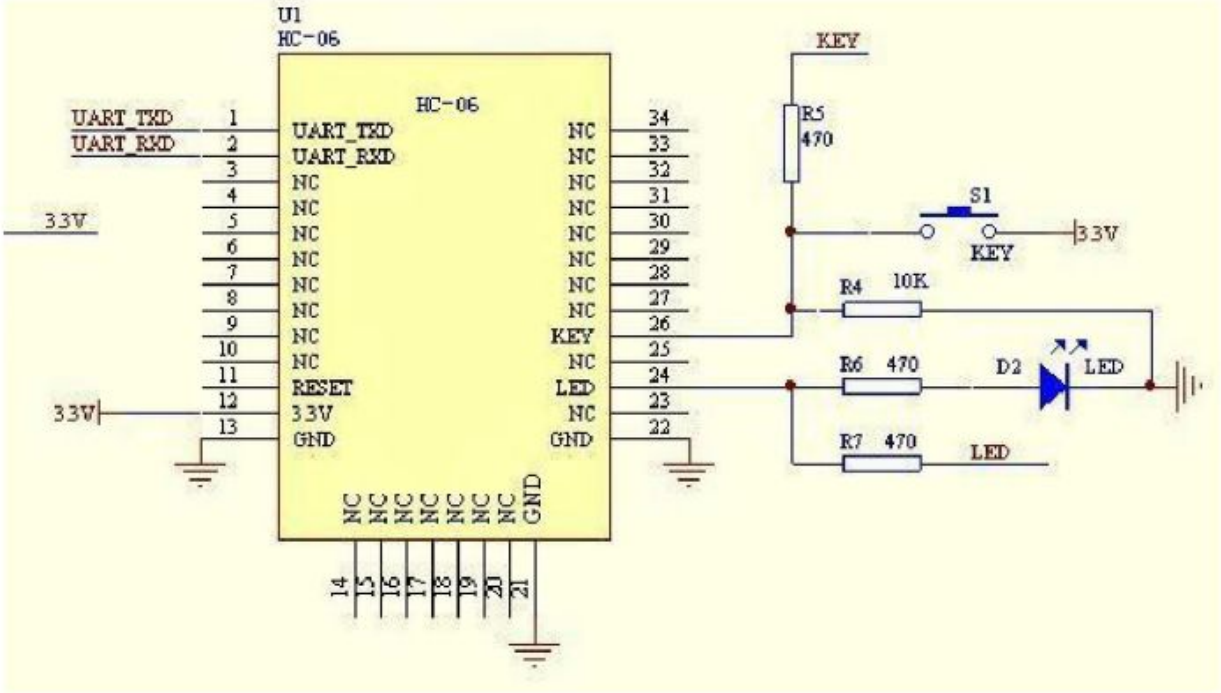
range (cm) = time (uS)/58




```
uint32_t P0_4 = (((0 << 12) | (4  << 8) | 0x30) & 0x0F00) >> 8;
uint32_t P0_3 = (((0 << 12) | (3  << 8) | 0x2c) & 0x0F00) >> 8;
while (1) {
	i=0;
	for(j=0;j<0xFF;j++);
	LPC_GPIO0->DATA &= ~(1<<7);                                    //trigger on
	if (LPC_GPIO0->MASKED_ACCESS[1 << P0_4] == 16) {          //check echo
		while(1){
			i++;
			if(LPC_GPIO0->MASKED_ACCESS[1 << P0_4] == 0) {
				//check if echo is low
				i = 0;
				break;
			}
		}
		if(i < 1000){
			//distance sensing
			//if dst is less than 1000 clocks, stop
			LPC_GPIO0->DATA &= ~(1<<3);
			LPC_GPIO0->DATA &= ~(1<<2);
			LPC_GPIO0->DATA |= (1<<8);
			LPC_GPIO0->DATA |= (1<<9);
			continue;
		}
		for(j=0;j<0x6FF;j++);                                  //wait
	}
```

Bluetooth

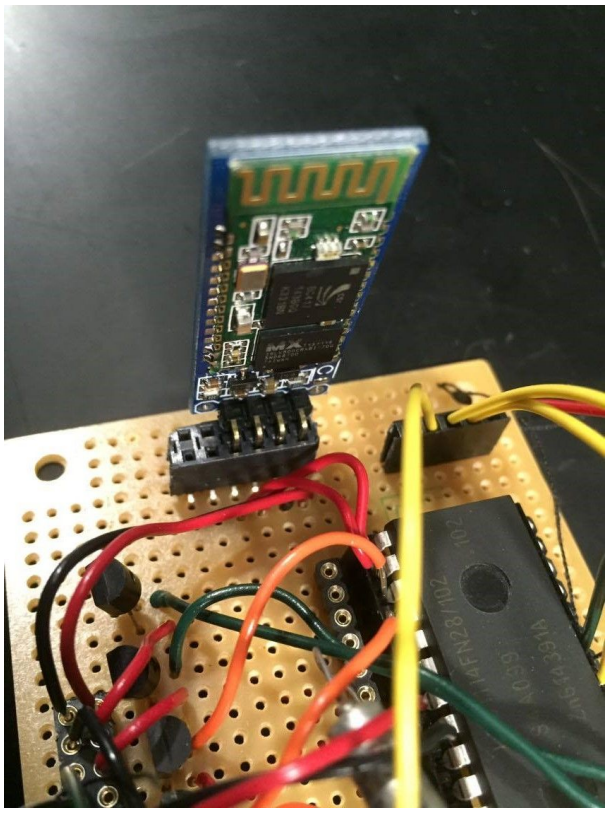
The HC-06 bluetooth module that we used for communicating is used for converting serial port to bluetooth. The HC-06 module is a slave module while the HC-05 module is a slave and master module which can connect to other slave devices. The main purpose of the bluetooth module is replacing serial input and converting it to the bluetooth standard so that it can communicate with another bluetooth device. The figure below shows how we connected up the module.




```
// Set up UART
void UART_init() {
    LPC_IOCON->PIO1_6             |= 0x01;      //configure UART RXD pin (sec 7.4.40)
    LPC_IOCON->PIO1_7             |= 0x01;      //configure UART TXD pin (sec 7.4.41)
    LPC_SYSCON->SYSAHBCLKCTRL     |= (1<<12);   //enable clock to UART (sec 3.5.14)
    LPC_SYSCON->UARTCLKDIV        |= 0x4e;      //set clk divider to 4e (sec 3.5.16)
    LPC_UART->FCR                 |= 0x01;      //enable FIFO (sec 13.5.6)
     LPC_UART->LCR                |= 0x03;      //set for 8 bit data width, 1 stop bit,
no parity (sec 13.5.7)
    LPC_UART->TER                 |= 0x80;      //enable transmission (sec 13.5.16)
}
```

After we setup the bluetooth module, we had to write code on the microcontroller side so that it could process the data that the bluetooth was passing onto it and had to write an android application that could send data to the bluetooth module. The android app is discussed below, so we will only discuss the serial processing for the microcontroller in this part. We discovered that since the bluetooth module translates all of the bluetooth format to serial, we only had to write a uart communications protocol for the microcontroller.

```
void UART_send(uint8_t data) {
	// Transmit data (sec 13.5.2)
	LPC_UART->THR |= data & 0xFF;
}
uint8_t UART_available() {
	// Overrun Error (sec 13.5.9)
	if (LPC_UART->LSR & BIT1) {
		return 16;
	} else {
		// Receiver Data Ready (sec 13.5.9)
		return LPC_UART->LSR & BIT0;
	}
}
uint8_t UART_receive() {
	// Receiver Buffer Register (sec 13.5.1)
	while(1){
		//wait for transmitted byte to loop back and be received
		if(LPC_UART->LSR & 0x01){
			//if Receiver Data Ready bit set (sec 13.5.9)
			Break;
		}
	}
	//store received data (sec 13.5.1)
	return LPC_UART->RBR;
}
```



This turned out to be fairly simplistic as the chip has registers which you can reference to see if there is any data waiting on the RX and TX lines and if there is, read it. The hardest part of writing the protocol was figuring out a bug where sometimes a bit more data would be read and sometimes nothing would be read. This turned out to be a problem with the baud rate of the device which we had to set. We had trouble with this for a while, since there were no good resources to help us with this. Finally, we found the arm documentation on github which had the directions on how to setup the baud rate for the chip. The code is provided below.

```
void serial_baud(int baudrate) {
    LPC_SYSCON->UARTCLKDIV = 0x1;
    uint32_t PCLK = SystemCoreClock;
    // First we check to see if the basic divide with no DivAddVal/MulVal
    // ratio gives us an integer result. If it does, we set DivAddVal = 0,
    // MulVal = 1. Otherwise, we search the valid ratio value range to find
    // the closest match. This could be more elegant, using search methods
    // and/or lookup tables, but the brute force method is not that much
    // slower, and is more maintainable.
    uint16_t DL = PCLK / (16 * baudrate);
```

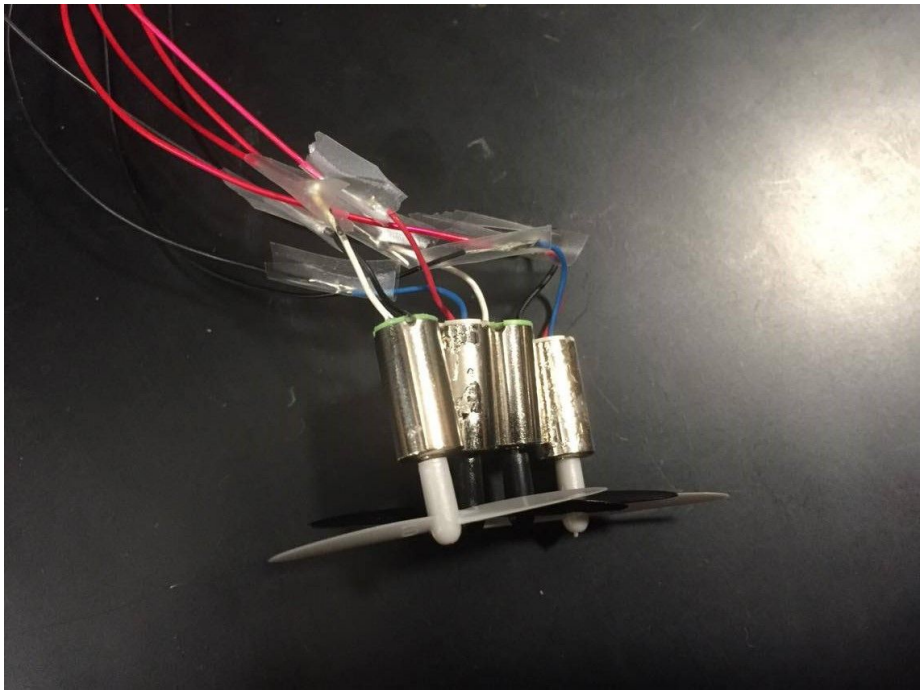
```
    uint8_t DivAddVal = 0;
    uint8_t MulVal = 1;
    int hit = 0;
    uint16_t dlv;
    uint8_t mv, dav;
    if ((PCLK % (16 * baudrate)) != 0) {     // Checking for zero remainder
        int err_best = baudrate, b;
        for (mv = 1; mv < 16 && !hit; mv++)
        {
            for (dav = 0; dav < mv; dav++)
            {
                if ((mv * PCLK * 2) & 0x80000000)
                    dlv = ((((2 * mv * PCLK) / (baudrate * (dav + mv))) / 16) + 1) /
2;
                else
                    dlv = ((((4 * mv * PCLK) / (baudrate * (dav + mv))) / 32) + 1) /
2;

                if (dlv == 0)
                    dlv = 1;

                // datasheet says if dav > 0 then DL must be >= 2
                if ((dav > 0) && (dlv < 2))
                    dlv = 2;

                // integer rearrangement of the baudrate equation (with rounding)
                b = ((PCLK * mv / (dlv * (dav + mv) * 8)) + 1) / 2;

                // check to see how we went
                b = abs(b - baudrate);
                if (b < err_best)
                {
                    err_best  = b;

                    DL        = dlv;
                    MulVal    = mv;
                    DivAddVal = dav;
```

```
                    if (b == baudrate)
                    {
                        hit = 1;
                        break;
                    }
                }
            }
        }
    }

    // set LCR[DLAB] to enable writing to divider registers
    LPC_UART->LCR |= (1 << 7);

    // set divider values
    LPC_UART->DLM = (DL >> 8) & 0xFF;
    LPC_UART->DLL = (DL >> 0) & 0xFF;
    LPC_UART->FDR = (uint32_t) DivAddVal << 0 | (uint32_t) MulVal    << 4;

    // clear LCR[DLAB]
    LPC_UART->LCR &= ~(1 << 7);
}
```

Motors

We used a total of four motors to drive the blimp. Two were used for going up, one for turning right, one for turning left, and the left and right motors together made it fly forward. The drive circuit was a fairly simplistic one: we had four outputs on the microcontroller, each of which were connected to the control pin of a bi-junctional transistor. The input of the transistor was a 3.3 voltage from one of the two voltage regulators. We realized that we needed two voltage regulators since one regulator could only supply 500 mA while each motor at max torque required around 450 mA. Therefore, we connected one up and left motor to one regulator and the other up and right motor to the other regulator. Since the up and forward motors would never be on at the same times, the max current draw through any regulator would not cap the current limit. The outputs of these transistors were connected to each respective motors and all the grounds of the motors were connected to the common ground.

The processing on the microcontroller's part was done via an if statement that processed through the message that the bluetooth module received. For example, if the bluetooth module sent a decimal '48' to the microcontroller via serial communication, the microcontroller would know this mean turn forward motors on and would turn the respective motor lines on and all other motor lines off. These motor lines would power the BJTs and they in turn would allow current to pass through to the motors.

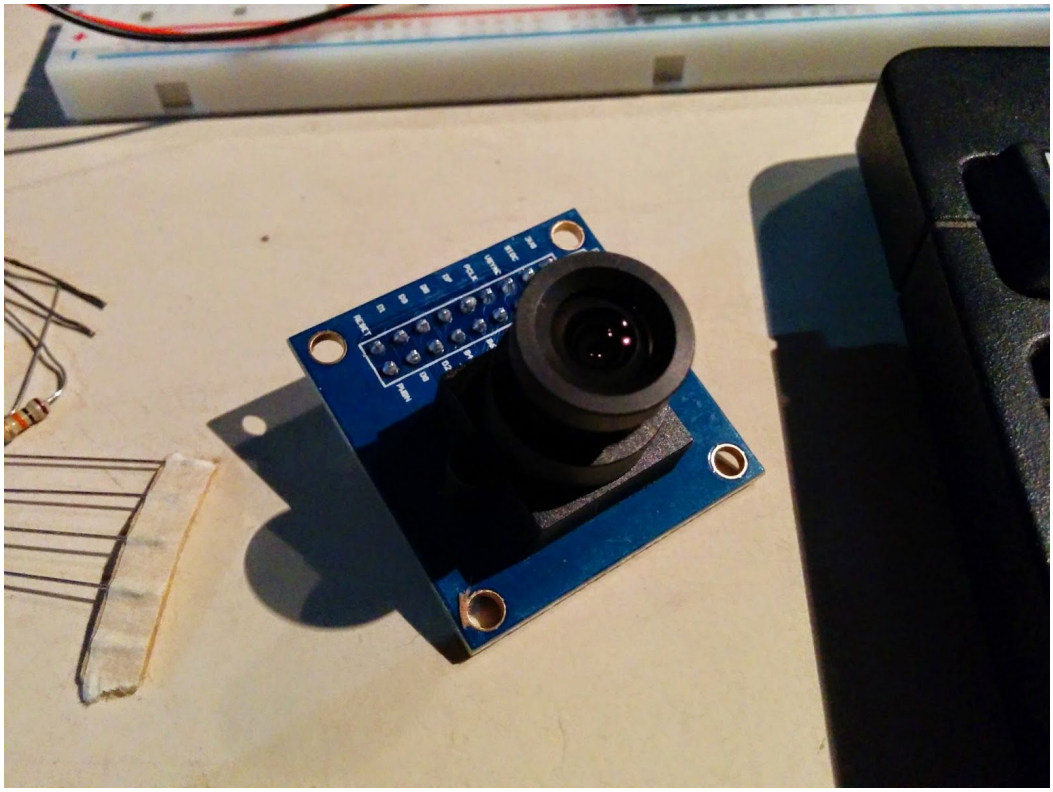


Camera

We spent a good three weeks to a month trying to get this to work. In the end, we had bytes of data streaming through serial, but we did not know if they were correct. The OV7670 has a lot of registers that need to be configured before the camera is actually useable and finding good resources on the camera is actually really hard. We have included the links that we found useful in the References section. The list of registers are also listed in the references section. `ov7670_set()` and `ov7670_get` are both wrappers around I2C set and get protocols.

```
void ov7670_init(void)
{
    printf("Initializing ov7670");

    printf("...settings");
    /*if (ov7670_get(REG_PID) != 0x76) {
        printf("PANIC! REG_PID != 0x76!\n");
        while (1);
    }*/
    ov7670_set(REG_COM7, 0x80); /* reset to default values */
    ov7670_set(REG_CLKRC, 0x80);
    ov7670_set(REG_COM11, 0x0A);
    ov7670_set(REG_TSLB, 0x04);
    ov7670_set(REG_TSLB, 0x04);
    ov7670_set(REG_COM7, 0x04); /* output format: rgb */

    ov7670_set(REG_RGB444, 0x00); /* disable RGB444 */
    ov7670_set(REG_COM15, 0xD0); /* set RGB565 */

    /* not even sure what all these do, gonna check the oscilloscope and go
     * from there... */
    ov7670_set(REG_HSTART, 0x16);
    ov7670_set(REG_HSTOP, 0x04);
    ov7670_set(REG_HREF, 0x24);
    ov7670_set(REG_VSTART, 0x02);
    ov7670_set(REG_VSTOP, 0x7a);
    ov7670_set(REG_VREF, 0x0a);
    ov7670_set(REG_COM10, 0x02);
    ov7670_set(REG_COM3, 0x04);
```

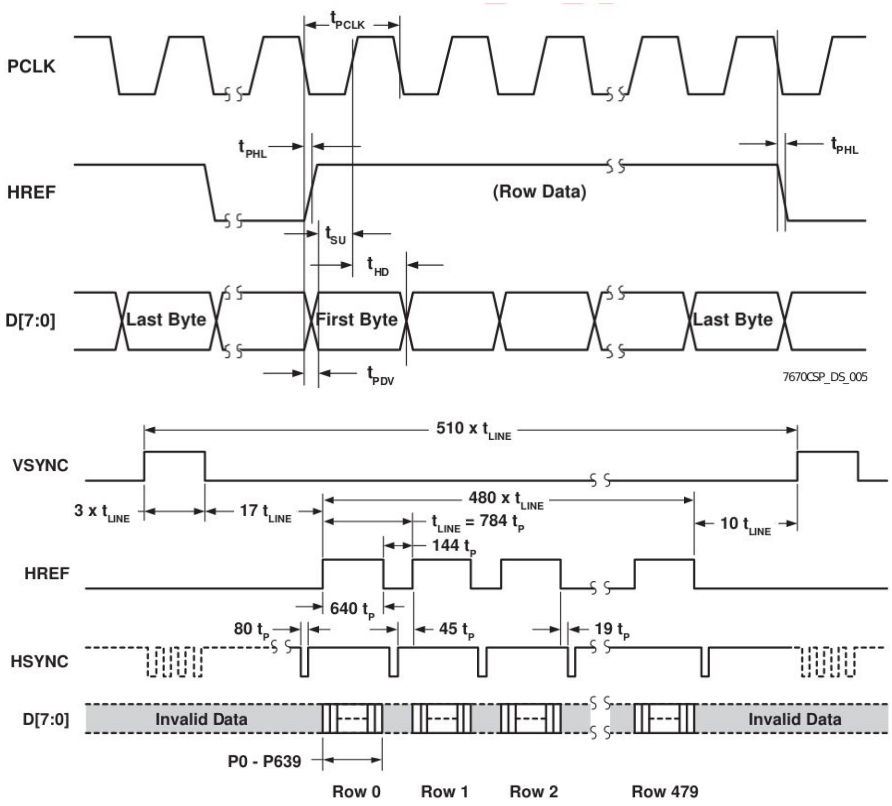
```
    ov7670_set(REG_COM14, 0x1a); // divide by 4
    //ov7670_set(REG_COM14, 0x1b); // divide by 8
    ov7670_set(REG_MVFP, 0x27);
    ov7670_set(0x72, 0x22); // downsample by 4
    //ov7670_set(0x72, 0x33); // downsample by 8
    ov7670_set(0x73, 0xf2); // divide by 4
    //ov7670_set(0x73, 0xf3); // divide by 8

    // test pattern
    //ov7670_set(0x70, 1 << 7);
    //ov7670_set(0x70, 0x0);

    // COLOR SETTING
    ov7670_set(0x4f, 0x80);
    ov7670_set(0x50, 0x80);
    ov7670_set(0x51, 0x00);
    ov7670_set(0x52, 0x22);
    ov7670_set(0x53, 0x5e);
    ov7670_set(0x54, 0x80);
    ov7670_set(0x56, 0x40);
    ov7670_set(0x58, 0x9e);
    ov7670_set(0x59, 0x88);
    ov7670_set(0x5a, 0x88);
    ov7670_set(0x5b, 0x44);
    ov7670_set(0x5c, 0x67);
    ov7670_set(0x5d, 0x49);
    ov7670_set(0x5e, 0x0e);
    ov7670_set(0x69, 0x00);
    ov7670_set(0x6a, 0x40);
    ov7670_set(0x6b, 0x0a);
    ov7670_set(0x6c, 0x0a);
    ov7670_set(0x6d, 0x55);
    ov7670_set(0x6e, 0x11);
    ov7670_set(0x6f, 0x9f);

    ov7670_set(0xb0, 0x84);

    printf("...done.\n");
}
```



To start off with, we realized that the camera VCC and other pins were supposed to driven at different voltages. The VCC is supposed to take anywhere from 2.6 V - 3.3 V, both another group burned their camera at 3.3 V so we recommend 3 V for VCC. To drive PCLK we recommend a peak-to-peak voltage of 2.7 V (what we used) and a driving frequency of 16 Mhz. This should (according to our calculations) give you about 3-5 frames per second. After this, your camera should be running and sending data out but you need to now make sense of the data. The timing diagrams below will tell you how and where to cut off reading data as it becomes garbage when HREF is low or VSYNC is is high. The HREF signal means that it is done reading a horizontal line of pixels while the VSYNC signal means that the camera is done reading the image.

```
void ov7670_readframe(void){
    while (ST_VSYNC); /* wait for the old frame to end */
    while (!ST_VSYNC); /* wait for a new frame to start */
    uint32_t i = 0;
    while (ST_VSYNC) {
        //if (y >= (QQVGA_HEIGHT / 2)) break;
        while (ST_VSYNC && !ST_HREF); /* wait for a line to start */
        if (!ST_VSYNC) break; /* line didn't start, but frame ended */
        while (ST_HREF) { /* wait for a line to end */
```

```
            /* first byte */
            while (!ST_PCLK); /* wait for clock to go high */
            /* no time to do anything fancy here! */
            /* this grabs the first 8 bits, rest gets chopped off */
             qqvgaframe1[i] = 0;
             qqvgaframe1[i] |= (ST_D7|ST_D6|ST_D5|ST_D4|ST_D3|ST_D2|ST_D1|ST_D0)
            while (ST_PCLK); /* wait for clock to go back low */
            i ++;
        }
    }
}
```



Once you are getting the pixel values, you have to realize that each byte is one-third of a pixel value and that these values are in YCbCr format and you have to convert them by using the formula shown below to RGB values.

$$
\begin{aligned}
R &= Y + 1.402 \cdot (C_R - 128) \\
G &= Y - 0.34414 \cdot (C_B - 128) - 0.71414 \cdot (C_R - 128) \\
B &= Y + 1.772 \cdot (C_B - 128)
\end{aligned}
$$

We also faced a problem of not having enough memory on the chip to store the entire 640x480 image (which is 921600 bytes), so decided to stream the bytes to the computer via serial as we were reading them. We don't know if this was the right way or even if got the correct values because we burned the camera before we could stitch the incoming values into an image.

In the end, we burned our camera by supplying 3 V signal too quickly; that is, we were touching the VCC, took the wire off the pin, and touched it again very quickly. There was no smoke and we are not quite sure why that burned the camera, but another group had burned four cameras before us so they are fragile and if we had to do it again, we would probably use another camera module.

Android Bluetooth Communication App

In order to communicate with our bluetooth slave-only module (HC-06), we decided to use an android NEXUS tablet and created an android application. This was because it was easier to pair with the module using the tablet when compared to a MAC or a PC. And also because android platform includes a lot of support for the bluetooth framework. Android provides support to perform the following operations.

- Scan for other devices
- Get a list of paired devices
- Connect to other devices through service discovery

It provides 'BluetoothAdapter' class to communicate with bluetooth. To create an object we use the static method 'getDefaultAdapter()'. And in order to enable the the bluetooth of our device we call the constant 'ACTION_REQUEST_ENABLE'. After we have enabled the bluetooth we can get a list of paired devices by using 'getBondedDevices()'. A part of our code is given below which shows the basic operation of the app.

```
void findBT() throws IOException
    {
        mBluetoothAdapter=BluetoothAdapter.getDefaultAdapter();
       // if(mBluetoothAdapter==null)
        //{
          //  myLabel.setText("No bluetooth adapter available");
        //}

        if(!mBluetoothAdapter.isEnabled())
        {
            Intent enableBluetooth=new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBluetooth, 1);
        }
        Set<BluetoothDevice> pairedDevices=mBluetoothAdapter.getBondedDevices();
        if(pairedDevices.size() > 0)
        {
            for(BluetoothDevice device : pairedDevices)
            {
                if(device.getName().equals(a))
                {
                    mmDevice=device;
                    break;
                }
            }
        }
```

Now, after we have established bluetooth connection we would need to send some data. Say we want to move 'front' and we want to send some stream we do this by creating an

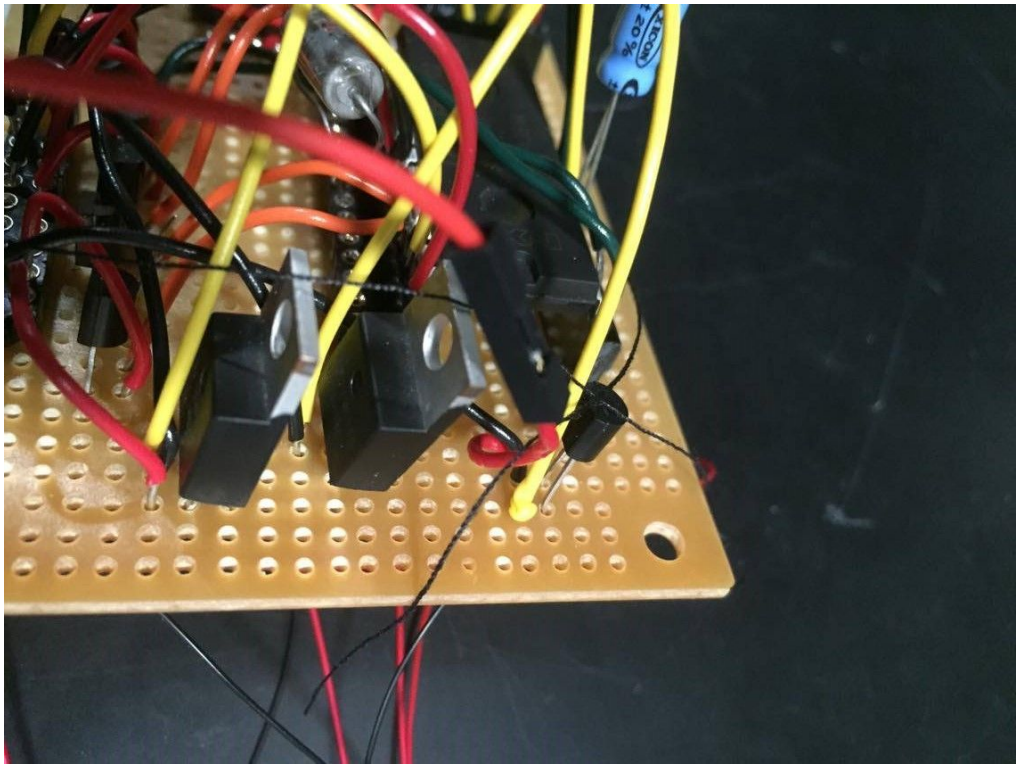
object of ‘OutputStream’ and use the ‘.write()’ function. Code snippet has been provided below. Here we keep sending a stream of ‘1’ until we press some other button.

```
Button b1 = (Button) findViewById(R.id.button1);// Front button
	b1.setOnClickListener(new View.OnClickListener() {

		@Override
		public void onClick(View v) {
			// TODO Auto-generated method stub
			try {
				mmOutputStream.write('1');
			} catch (IOException e) {
				// TODO Auto-generated catch block
				Toast.makeText(Menuuu.this, "Connection not established
with the robot", Toast.LENGTH_LONG).show();
				e.printStackTrace();
			}

			}
	});
```

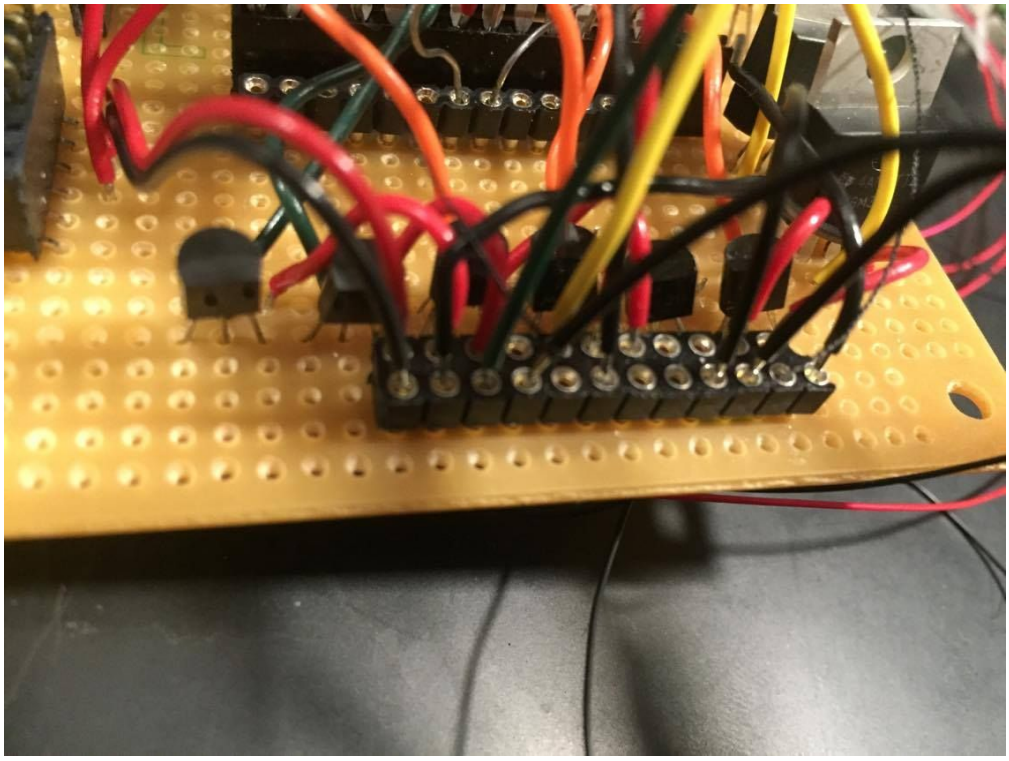
We follow the same procedure for all other buttons. After we have created the layout file and the main activity xml file. There is android app development tutorial and advance tips for creating app available on lynda.com (free if you are UIUC student) that describe the basics and teach android development from scratch.

Soldering

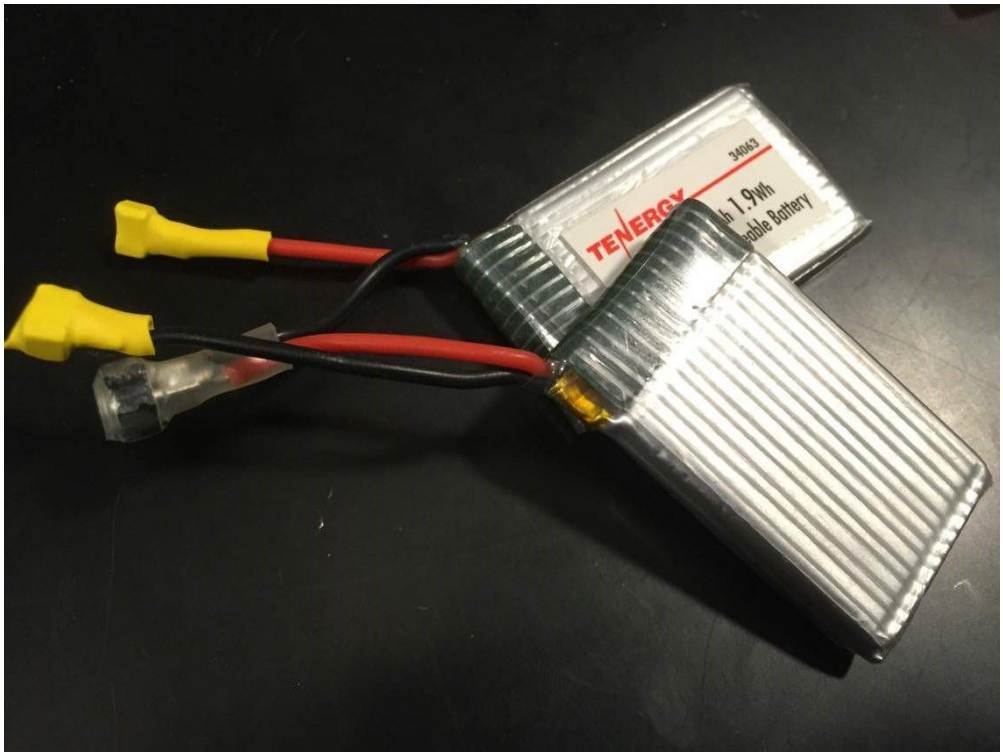
Soldering the circuit together had been a crucial component for our blimp, because a mini breadboard was too heavy, and PCB's take a while to make. We came across a few problems. First, we started by soldering the LPC1114 chip directly to the board. This was very risky. Its because soldering heats up the electrical components, and as a result we broke the chip. Hence we decided to go with header pins. We also used them to solder the power and ground lines as well. After soldering the voltage regulators, power lines, the BJT's and batteries together we came across a problem. After we turned on a single motor, all the pins became high in the LPC1114 chip. We and the motor wouldn't stop. We figured that the problem was we needed two 3.3V voltage regulators as the max current supply from one of them couldn't handle all 4 motors. Also that we shouldn't supply the input to the 3.3V from the output of the 5V regulator. After fixing these problems the circuit worked perfectly.



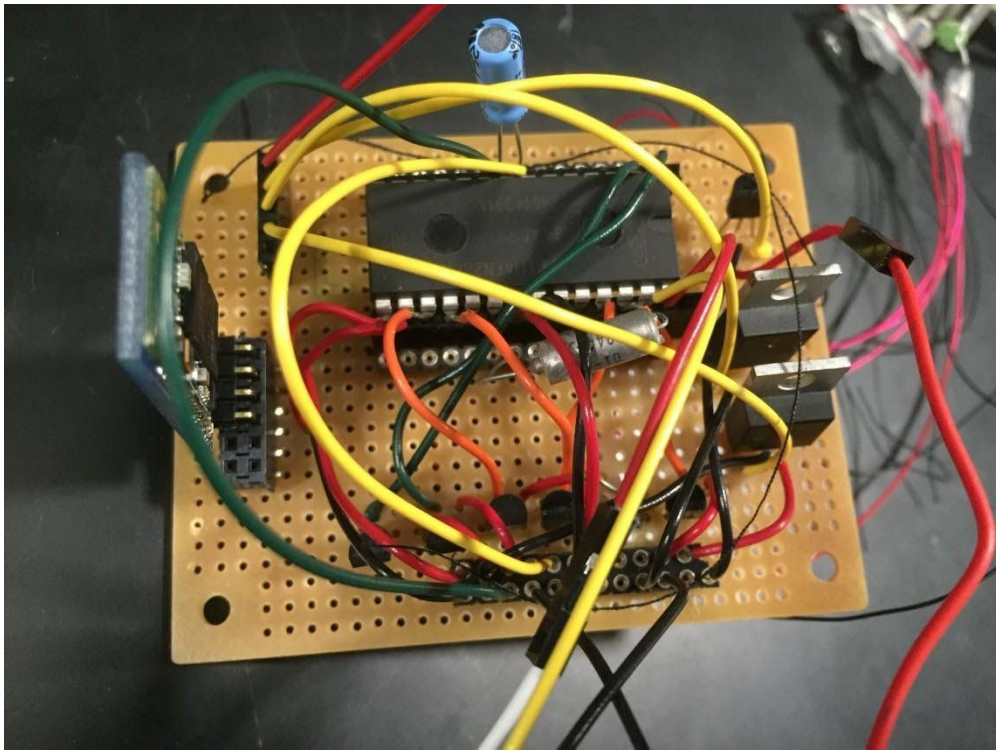
The two 3.3 V and one 5 V voltage regulators



The power and ground lines



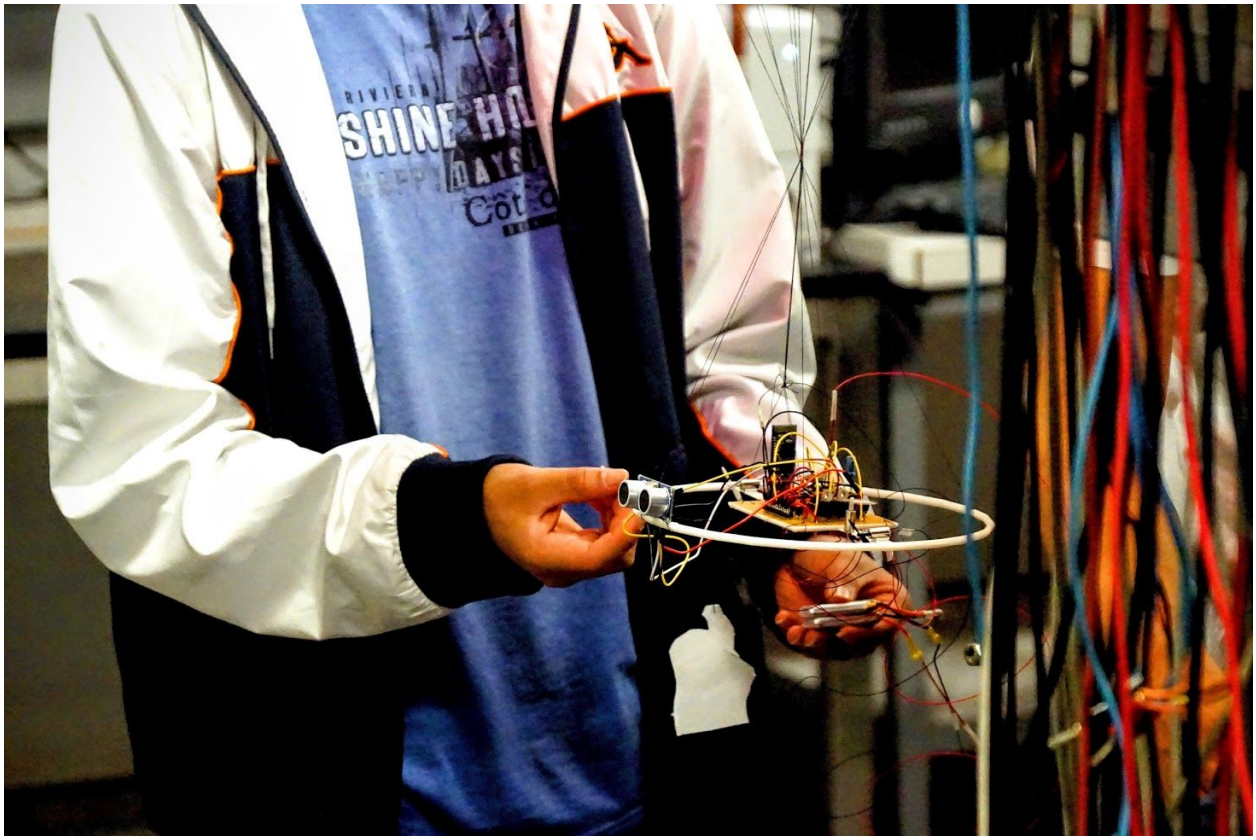
Soldering the batteries together to get 7.4 V



The entire circuit

Results

We got the blimp hovering with eight 9-inch balloons filled with helium. We twisted all the balloon strings together in the center so that the weight would be evenly distributed and the blimp would not tilt in any one direction. The frame weighed 14 grams and the total weight excluding the batteries was about 120 grams. We held the batteries in our hands as the frame could no support them because it was structurally weak, and we would have to add a lot more balloons to support the entire thing. Everything worked and the blimp could go forward, upward, turn right and left and stop when it detected an obstacle.



Once we had everything running, we realized that since we did not have a control system or speed control, once you turned the blimp a certain way, say right, it would keep turning right even after you stopped turning right and would tangle all the strings and wires together. We also had to add a counterweight in the back since the distance sensor kept tipping the blimp forward.

Future work

For future work, we would and plan on getting the camera working and streaming the bytes to an android device that can do image processing and send back directions that the blimp or quadcopter (whichever we are working with at that time) could follow autonomously to follow a target. For this we would need either a more powerful chip or we need a better way transmit video data to Android.

Other than that, we realized that the blimp definitely needs some form of control system to stop it from going crazy when the motors start and stop turning. For that we would need sensors that give us some form of feedback in a closed loop. We would also need a gyroscope or accelerometer that would tell the microcontroller if the tilt and the speed of each motor changes and if it goes out of bound it automatically stabilizes itself. That would give us a complete ‘Follow-me Blimp’.

References

A. List of registers for OV7670

```
#define REG_GAIN        0x00    /* Gain lower 8 bits (rest in vref) */
#define REG_BLUE        0x01    /* blue gain */
#define REG_RED         0x02    /* red gain */
#define REG_VREF        0x03    /* Pieces of GAIN, VSTART, VSTOP */
#define REG_COM1        0x04    /* Control 1 */
#define COM1_CCIR656    0x40    /* CCIR656 enable */
#define REG_BAVE        0x05    /* U/B Average level */
#define REG_GbAVE       0x06    /* Y/Gb Average level */
#define REG_AECHH       0x07    /* AEC MS 5 bits */
#define REG_RAVE        0x08    /* V/R Average level */
#define REG_COM2        0x09    /* Control 2 */
#define COM2_SSLEEP     0x10    /* Soft sleep mode */
#define REG_PID         0x0a    /* Product ID MSB */
#define REG_VER         0x0b    /* Product ID LSB */
#define REG_COM3        0x0c    /* Control 3 */
#define COM3_SWAP       0x40    /* Byte swap */
#define COM3_SCALEEN    0x08    /* Enable scaling */
#define COM3_DCWEN      0x04    /* Enable downsamp/crop/window */
#define REG_COM4        0x0d    /* Control 4 */
#define REG_COM5        0x0e    /* All "reserved" */
#define REG_COM6        0x0f    /* Control 6 */
#define REG_AECH        0x10    /* More bits of AEC value */
#define REG_CLKRC       0x11    /* Clocl control */
#define CLK_EXT         0x40    /* Use external clock directly */
#define CLK_SCALE       0x3f    /* Mask for internal clock scale */
#define REG_COM7        0x12    /* Control 7 */
#define COM7_RESET      0x80    /* Register reset */
#define COM7_FMT_MASK   0x38
#define COM7_FMT_VGA    0x00
#define COM7_FMT_CIF    0x20    /* CIF format */
#define COM7_FMT_QVGA   0x10    /* QVGA format */
#define COM7_FMT_QCIF   0x08    /* QCIF format */
#define COM7_RGB        0x04    /* bits 0 and 2 - RGB format */
#define COM7_YUV        0x00    /* YUV */
#define COM7_BAYER      0x01    /* Bayer format */
#define COM7_PBAYER     0x05    /* "Processed bayer" */
```

```
#define REG_COM8        0x13    /* Control 8 */
#define COM8_FASTAEC    0x80    /* Enable fast AGC/AEC */
#define COM8_AECSTEP    0x40    /* Unlimited AEC step size */
#define COM8_BFILT      0x20    /* Band filter enable */
#define COM8_AGC        0x04    /* Auto gain enable */
#define COM8_AWB        0x02    /* White balance enable */
#define COM8_AEC        0x01    /* Auto exposure enable */
#define REG_COM9        0x14    /* Control 9  - gain ceiling */
#define REG_COM10       0x15    /* Control 10 */
#define COM10_HSYNC     0x40    /* HSYNC instead of HREF */
#define COM10_PCLK_HB   0x20    /* Suppress PCLK on horiz blank */
#define COM10_HREF_REV  0x08    /* Reverse HREF */
#define COM10_VS_LEAD   0x04    /* VSYNC on clock leading edge */
#define COM10_VS_NEG    0x02    /* VSYNC negative */
#define COM10_HS_NEG    0x01    /* HSYNC negative */
#define REG_HSTART      0x17    /* Horiz start high bits */
#define REG_HSTOP       0x18    /* Horiz stop high bits */
#define REG_VSTART      0x19    /* Vert start high bits */
#define REG_VSTOP       0x1a    /* Vert stop high bits */
#define REG_PSHFT       0x1b    /* Pixel delay after HREF */
#define REG_MIDH        0x1c    /* Manuf. ID high */
#define REG_MIDL        0x1d    /* Manuf. ID low */
#define REG_MVFP        0x1e    /* Mirror / vflip */
#define MVFP_MIRROR     0x20    /* Mirror image */
#define MVFP_FLIP       0x10    /* Vertical flip */
#define REG_AEW         0x24    /* AGC upper limit */
#define REG_AEB         0x25    /* AGC lower limit */
#define REG_VPT         0x26    /* AGC/AEC fast mode op region */
#define REG_HSYST       0x30    /* HSYNC rising edge delay */
#define REG_HSYEN       0x31    /* HSYNC falling edge delay */
#define REG_HREF        0x32    /* HREF pieces */
#define REG_TSLB        0x3a    /* lots of stuff */
#define TSLB_YLAST      0x04    /* UYVY or VYUY - see com13 */
#define REG_COM11       0x3b    /* Control 11 */
#define COM11_NIGHT     0x80    /* NIght mode enable */
#define COM11_NMFR      0x60    /* Two bit NM frame rate */
#define COM11_HZAUTO    0x10    /* Auto detect 50/60 Hz */
#define COM11_50HZ      0x08    /* Manual 50Hz select */
#define COM11_EXP       0x02
```

```
#define REG_COM12       0x3c    /* Control 12 */
#define COM12_HREF      0x80    /* HREF always */
#define REG_COM13       0x3d    /* Control 13 */
#define COM13_GAMMA     0x80    /* Gamma enable */
#define COM13_UVSAT     0x40    /* UV saturation auto adjustment */
#define COM13_UVSWAP    0x01    /* V before U - w/TSLB */
#define REG_COM14       0x3e    /* Control 14 */
#define COM14_DCWEN     0x10    /* DCW/PCLK-scale enable */
#define REG_EDGE        0x3f    /* Edge enhancement factor */
#define REG_COM15       0x40    /* Control 15 */
#define COM15_R10F0     0x00    /* Data range 10 to F0 */
#define COM15_R01FE     0x80    /*            01 to FE */
#define COM15_R00FF     0xc0    /*            00 to FF */
#define COM15_RGB565    0x10    /* RGB565 output */
#define COM15_RGB555    0x30    /* RGB555 output */
#define REG_COM16       0x41    /* Control 16 */
#define COM16_AWBGAIN   0x08    /* AWB gain enable */
#define REG_COM17       0x42    /* Control 17 */
#define COM17_AECWIN    0xc0    /* AEC window - must match COM4 */
#define COM17_CBAR      0x08    /* DSP Color bar */
#define REG_CMATRIX_BASE 0x4f
#define CMATRIX_LEN 6
#define REG_CMATRIX_SIGN 0x58
#define REG_BRIGHT      0x55    /* Brightness */
#define REG_CONTRAS     0x56    /* Contrast control */
#define REG_GFIX        0x69    /* Fix gain control */
#define REG_REG76       0x76    /* OV's name */
#define R76_BLKPCOR     0x80    /* Black pixel correction enable */
#define R76_WHTPCOR     0x40    /* White pixel correction enable */
#define REG_RGB444      0x8c    /* RGB 444 control */
#define R444_ENABLE     0x02    /* Turn on RGB444, overrides 5x5 */
#define R444_RGBX       0x01    /* Empty nibble at end */
#define REG_HAECC1      0x9f    /* Hist AEC/AGC control 1 */
#define REG_HAECC2      0xa0    /* Hist AEC/AGC control 2 */
#define REG_BD50MAX     0xa5    /* 50hz banding step limit */
#define REG_HAECC3      0xa6    /* Hist AEC/AGC control 3 */
#define REG_HAECC4      0xa7    /* Hist AEC/AGC control 4 */
#define REG_HAECC5      0xa8    /* Hist AEC/AGC control 5 */
#define REG_HAECC6      0xa9    /* Hist AEC/AGC control 6 */
```

```
#define REG_HAECC7      0xaa    /* Hist AEC/AGC control 7 */
#define REG_BD60MAX     0xab    /* 60hz banding step limit */
```

B. List of useful websites
    a. Camera
        i. http://embeddedprogrammer.blogspot.com/2012/07/hacking-ov7670-camera-module-sccb-cheat.html
        ii. http://www.elecfreaks.com/projects/how-to-use-ov7670-camera-module-with-arduino/
        iii. http://web.mit.edu/6.111/www/f2015/tools/OV7670_2006.pdf
        iv. http://www.rpg.fi/desaster/blog/2012/05/07/ov7670-camera-sensor-success/
    b. Distance Sensor
        i. http://www.micropik.com/PDF/HCSR04.pdf
    c. Bluetooth
        i. http://www.tec.reutlingen-university.de/uploads/media/DatenblattHC-05_BT-Modul.pdf
        ii. http://elecfreaks.com/store/download/datasheet/Bluetooth/HC-06-Spec.pdf
        iii. https://eewiki.net/display/microcontroller/Getting+Started+with+NXP's+LPC11XX+Cortex-M0+ARM+Microcontrollers#GettingStartedwithNXP'sLPC11XXCortex-M0ARMMicrocontrollers-CodeExample0:ToggleaGPIOPin
    d. Android app development
        i. https://www.lynda.com/Android-tutorials/Manage-Gradle-build-scripts/442863/456759-4.html
        ii. http://developer.android.com/guide/topics/connectivity/bluetooth.html