**Simple Home**

**Surveillance**

**CpE 301 – 1001**

**Final Project**

**May 4, 2017**

**Team #15**

**Aaron Volpone**

***Abstract*** – the goal of this project is to create a cheap, simple, and discreet surveillance package for a home user. Though just a prototype, the idea was to test the notion that it is plausible to compete with similarly priced surveillance cameras on the market with individually sourced parts. With a limited budget, this pertains to accessible hobbyist hardware across the web.

1. **Introduction**

The growth of IoT (Internet of Things) products across all walks of daily life have proven to be an exciting and lucrative avenue for engineers. For a home consumer, microprocessor items exist in nearly every electronic product in the house. Kitchen appliances, home entertainment systems, and mobile devices. Yet another form of IoT exists in home security systems like house alarm management and surveillance. The idea of this engineering project is to emulate building a network connected surveillance package with limited money and time. The flow of development is to start from the ground up to with fundamental functionality and work into more complex topics like wireless connectivity, notifications, and even motion tracking.

Unfortunately, due to time constraints, not all stretch goals were able to be accomplished within the given time and budget. With just basic functionality alone, over $60 is spent just to get a functioning model. I categorized wireless connectivity as a stretch goal and used the remaining time to structure the core aspect of the camera package. That is, enabling an easy way to capture pictures and store them in an automated manner. The prototype possesses the capability to take pictures when detecting motion (toggled on or off by the user) and pivot with a joystick.

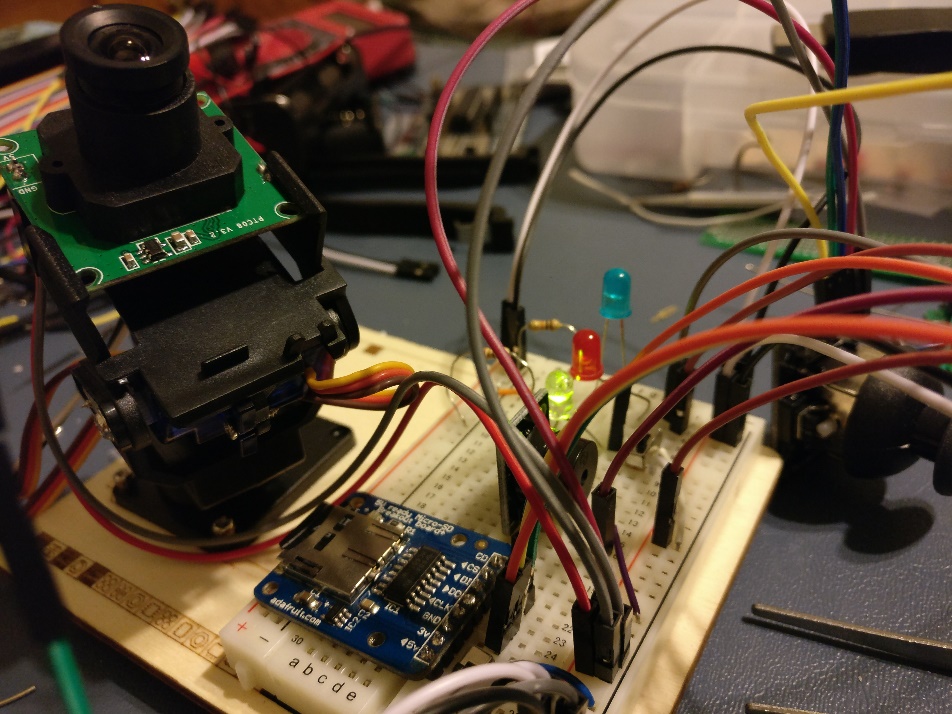
1. **Implementation**

To fit within a reasonable overall price tag for the project, it became apparent to acquire easily accessible hobbyist equipment to kick start the project. This included shopping with the awareness of my limited knowledge of advanced AVR applications and data transmission protocols. Since working with serial transmission through lecture and lab, I became a familiar with the topic out in the real world with various kinds of hobbyist hardware.

The primary item for the project focuses on the selected camera for operation. Ultimately, TTL serial cameras became the only option for the sake of familiarity and ease of use. The camera selected camera chipset model, VC0607, wired to the ATmega the same way as a UART communication bridge is made. That is, data is transferred through the TX and RX pins. My first running prototype built the camera on the Uno until I was ready to migrate my library modifications changes onto a plain ATmega328P chip. Complications that arose from this act are explained later in the document. Despite software complications, the camera operates still manages to operate rather abstractly since commands are natively programmed into the camera chipset.

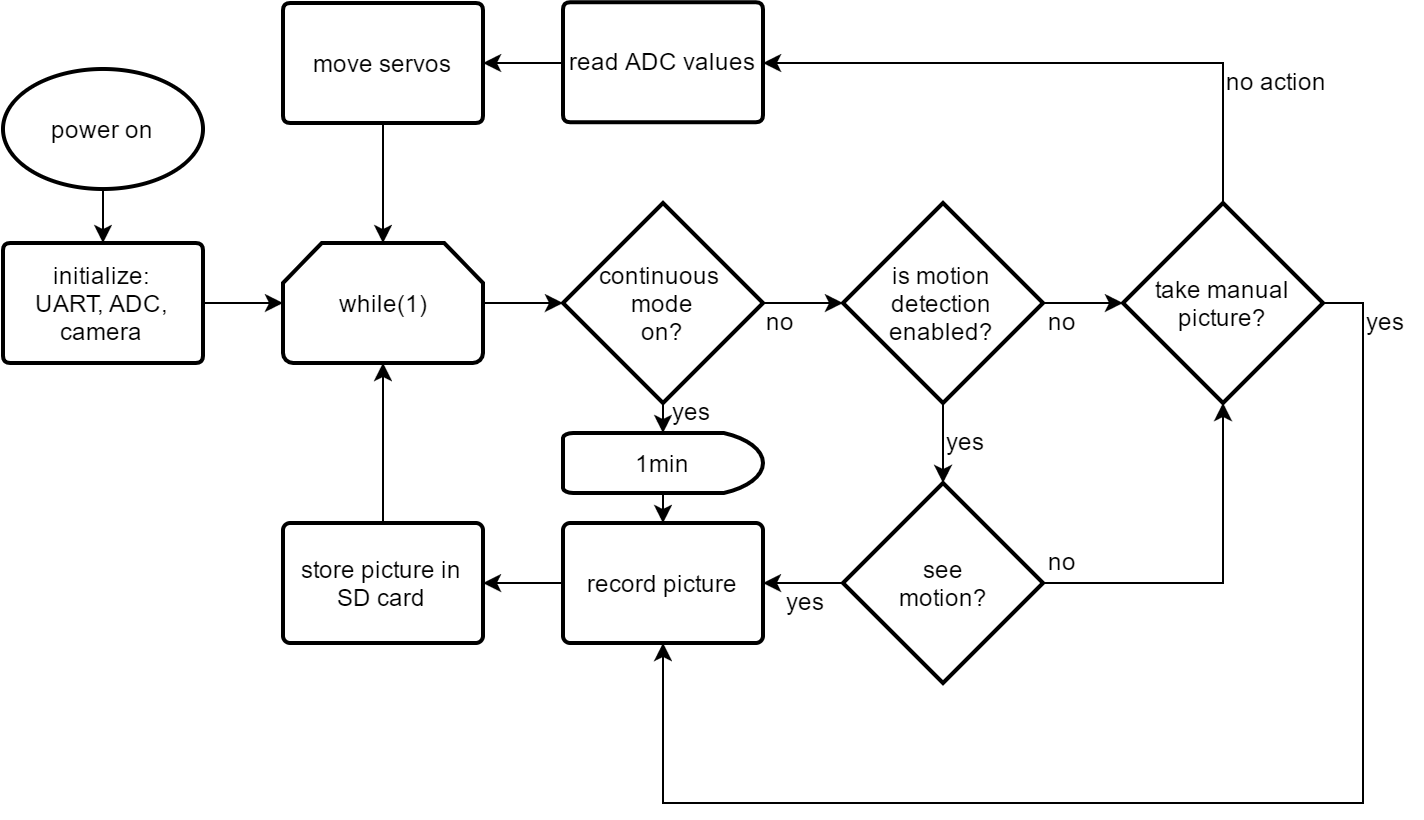
To save captured photos, a method of storage is needed. The memory card breakout item I chose for my project runs on a low 3.3V input and follows SD card file storage protocol. Additionally, it uses a MicroSD form factor card, thus minimizing the profile of the breadboard prototype. Implementing the libraries for SD protocol are, like the camera, highly abstracted and made troubleshooting quite an issue when running into file storage issues. However, getting the camera and card to finally communicate contributed to a big portion of the overall project’s weight. So making sure they work as intended is necessary.

The next item for basic functionality is made possible with the implementation of the tilt-swivel stand that the camera sits on. Adafruit provides easy startup materials for enabling animation of mechanical hardware for projects like cars and robots. I chose to use a dual micro servo mount that operated very easily through the PWM pins provided by the ATmega. Specifically, a 10-bit fast PWM timer is used for both servos. This enables quick yet minute adjustments necessary to orient the camera effectively through an analog joystick. Theoretically, the implementation of motion tracking can enable very precise adjustments since the value range of the timers are so large for the servos (1023-bit range) – a welcoming capability when motion tracking can be added.

Remaining items for the surveillance package exist in the form of LED lights and a buzzer. These are implemented to provide the end-user a very basic means of informing them the status of the overall surveillance system. All modules including the camera and SD card reader are shown below:

The green, red, and blue LED are responsible for notifying the user of various states of the package. Motion detection mode is shown on in the above picture since the green LED is lit. Additionally if there is an error, the program halts and leaves the package with a red LED on. Lastly, when the camera is processing picture data (reading/writing), the blue LED is active. The existing buzzer beeps correspondingly depending on these various states indicated by the LED.

To fill in missing connectivity features provided by a wireless connection, I accomplished an alternative goal of creating a UART bridge for the camera. This would allow the interfacing of the surveillance package to an appropriate computing platform – enabling the user to see the status of the camera and whether a picture is currently being processed.



1. **Experimental evaluation / Results**

Initial testing begins with the surrounding infrastructure of the camera package. That is, ensuring the proper operation of the joystick and micro servos for camera orientation. Since the servos

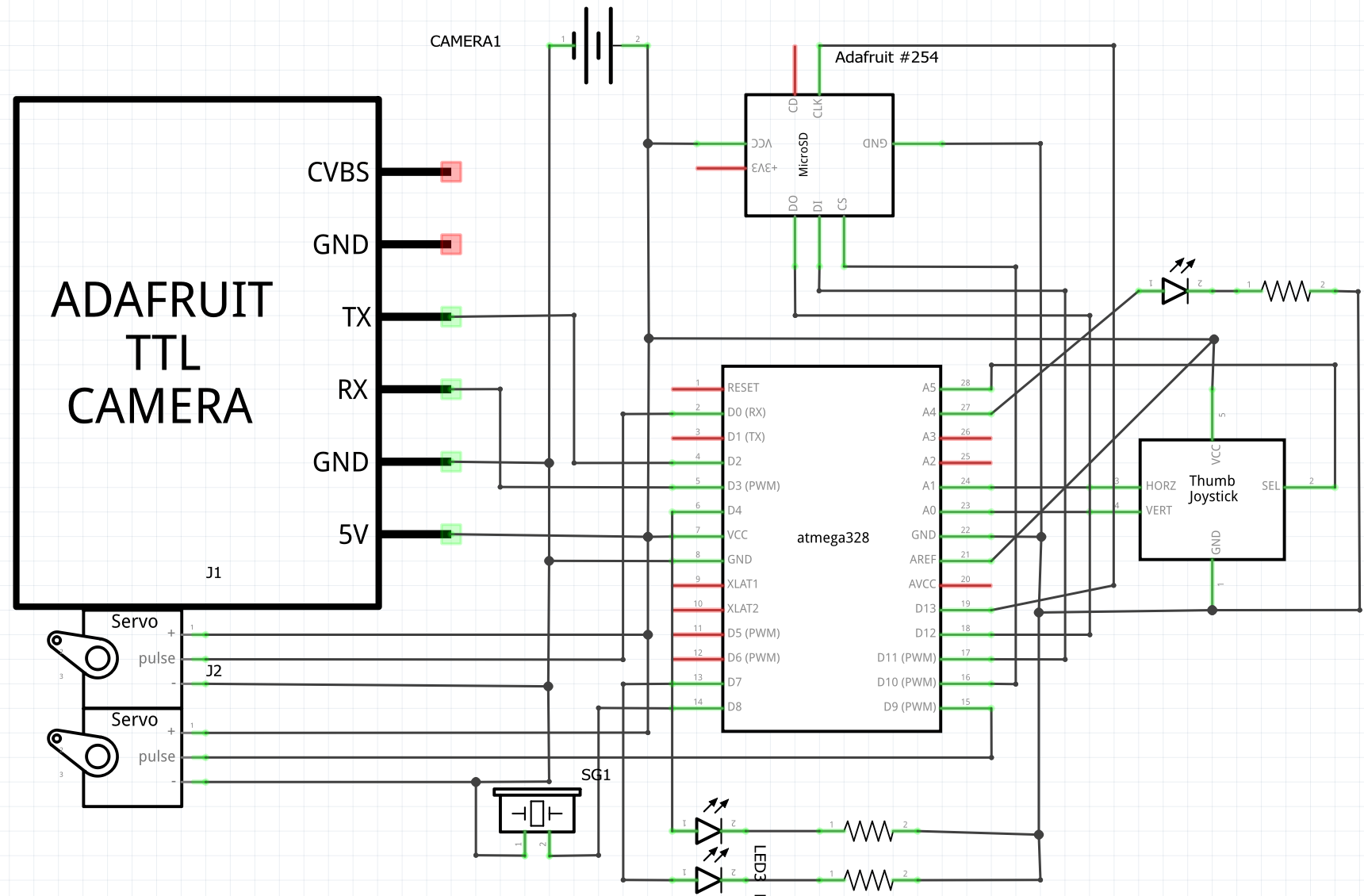
Describe how you evaluated the system’s performance. This section should provide examples that highlight what input devices and sequence of actions you used to affect a result and how those results can be observed and confirmed. There should be sample input signals and corresponding output on an output device.

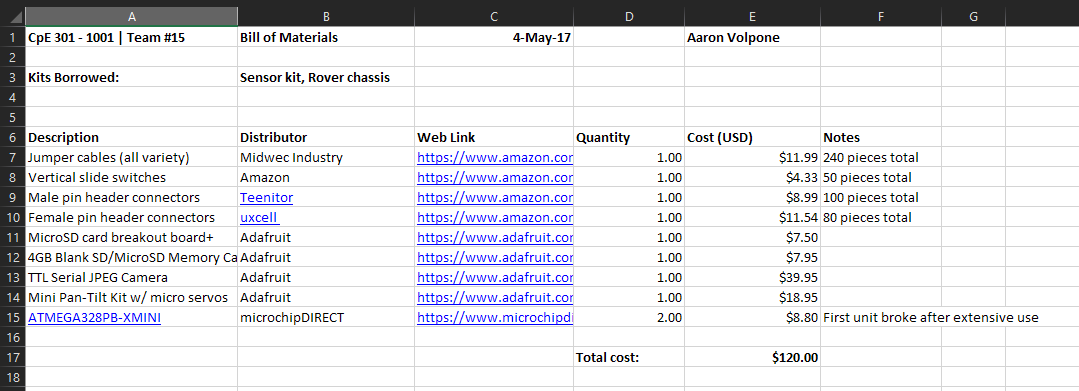
1. **Conclusion**

The conclusion of the report should provide a summary of the project aim and results as well as highlight both what was learned working on the project and what further directions would improve the project.

1. **Appendix**

* **Detailed Schematics**
* **Bill of Materials**
* **Code:** Well-commented, organized code





* **Video Demonstration**

<https://www.youtube.com/watch?v=qWdbE6uHGSE>

* **Main driver file**

**#define F\_CPU 16000000UL // CPU clock speed**

**// C standard library**

**#include <stdlib.h>**

**// AVR related libraries**

**#include <avr/io.h>**

**#include <avr/interrupt.h>**

**#include <util/delay.h>**

**#include <avr/power.h>**

**#include <avr/sleep.h>**

**#include <ioe.c> // personal UART library**

**#define ARDUINO 100 // spoof Arduino libraries with board version**

**#define chipSelect 10 // pin for card selection**

**#define TRUE 1**

**#define FALSE 0**

**#define NORMAL 0**

**#define MOTION 1**

**#define LAPSE 2**

**#define COMBINED 3**

**// Arduino related libraries**

**#include <Arduino.h>**

**// Software serial library**

**#include <SoftwareSerial.h>**

**// Camera libraries**

**#include <Adafruit\_VC0706.h>**

**#include <SPI.h>**

**// SD card library**

**#include <SD.h>**

**// create a connection through a software serial method**

**SoftwareSerial cameraconnection = SoftwareSerial(5, 6); // 2, 3**

**Adafruit\_VC0706 cam = Adafruit\_VC0706(&cameraconnection);**

**// function prototypes**

**void ADC\_init();**

**void cam\_init();**

**void init\_fail();**

**void init\_pass();**

**void cam\_snap();**

**void cam\_checkMode();**

**unsigned int SDcheck = FALSE;**

**unsigned int reset = TRUE;**

**unsigned int menu = FALSE;**

**unsigned int opMode = NORMAL;**

**unsigned char input;**

**char voltageStr[20];**

**unsigned int ADCV = 0;**

**float voltage = 0; // temporary variable for holding conversion**

**void setup()**

**{**

**if(reset)**

**{**

**UART\_init();**

**ADC\_init(); // initialize ADC**

**cam\_init();**

**reset = FALSE;**

**}**

**}**

**void loop()**

**{**

**ADC\_init();**

**// ==================================================**

**// X-AXIS JOYSTICK**

**// ==================================================**

**ADMUX = (1 << MUX0); // check ADC port 2**

**ADCSRA |= (1 << ADSC) | (0 << REFS0) | (0 << REFS1); // start conversion, relative to Vcc**

**while( (ADCSRA & (1 << ADIF) ) == 0 ); // wait for conversion to finish**

**ADCV = ADC & 1023; // grab only the 10 bits**

**voltage = ( (ADCV \* 5) / 1023); // convert int to float**

**/\***

**dtostrf(voltage, 4, 3, voltageStr); // float to string**

**UART\_putStr(voltageStr); // print the voltage**

**UART\_putStr(" V [X-AXIS] \n\r"); // print voltage suffix**

**Serial.println(OCR1A); // optionally, print servo position**

**UART\_putStr("\n\r"); // line break**

**\*/**

**// x-axis servo logic**

**if(voltage < 1) // if ADC lower than median joystick voltage**

**{**

**if(OCR1A >= 600) // check if upper servo position maxed out**

**OCR1A = 600; // if so, reset to max**

**else**

**OCR1A += 3; // else, increase position by 3**

**\_delay\_ms(15); // delay for speed reduction**

**}**

**else if(voltage > 3) // else if ADC higher than median joystick voltage**

**{**

**if(OCR1A <= 150) // check if lower servo position maxed out**

**OCR1A = 150; // if so, reset to min**

**else**

**OCR1A -= 3; // else, decrease position by 3**

**\_delay\_ms(15); // delay for speed reduction**

**}**

**// ==================================================**

**// Y-AXIS JOYSTICK**

**// ==================================================**

**ADMUX = 0;**

**ADCSRA |= (1 << ADSC) | (0 << REFS0) | (0 << REFS1);**

**while( (ADCSRA&(1<<ADIF)) == 0 ); // wait for conversion**

**ADCV = ADC & 0x03FF; // grab only the 10 bits**

**voltage = (ADCV \* 5.0 / 0x3FF); // convert int to float**

**/\***

**dtostrf(voltage, 4, 3, voltageStr); // convert float to int**

**UART\_putStr(voltageStr); // print theSS voltage**

**UART\_putStr(" V [Y-AXIS] \n\r"); // suffix**

**Serial.println(OCR4B);**

**UART\_putStr("\n\r"); // suffix**

**\*/**

**if(voltage < 1)**

**{**

**if(OCR3B >= 500 || OCR4B >= 500)**

**{**

**OCR3B = 500;**

**OCR4B = 500;**

**}**

**else**

**{**

**OCR3B += 3;**

**OCR4B += 3;**

**}**

**\_delay\_ms(15);**

**}**

**else if(voltage > 3)**

**{**

**if(OCR3B <= 200 || OCR4B <= 200)**

**{**

**OCR3B = 200;**

**OCR4B = 200;**

**}**

**else**

**{**

**OCR3B -= 3;**

**OCR4B -= 3;**

**}**

**\_delay\_ms(15);**

**}**

**// ==================================================**

**// JOYSTICK BUTTON**

**// ==================================================**

**ADMUX = (0 << MUX3) | (1 << MUX2) | (0 << MUX1) | (1 << MUX0)**

**| (0 << REFS0) | (0 << REFS1);**

**ADCSRA |= (1 << ADSC);**

**while( (ADCSRA&(1<<ADIF)) == 0 ); // wait for conversion**

**ADCV = ADC & 0x03FF; // grab only the 10 bits**

**voltage = (ADCV \* 5.0 / 0x3FF); // convert int to float**

**/\***

**dtostrf(voltage, 4, 3, voltageStr); // convert float to int**

**UART\_putStr(voltageStr); // print theSS voltage**

**UART\_putStr(" V [BUTTON] \n\r"); // suffix**

**UART\_putStr("\n\r"); // suffix**

**\*/**

**if(voltage < 0.040 && voltage > 0.030)**

**{**

**\_delay\_ms(15);**

**cam\_snap();**

**\_delay\_ms(15);**

**}**

**if(opMode == MOTION || opMode == COMBINED )**

**{**

**if(cam.motionDetected())**

**cam\_snap();**

**}**

**if(opMode == LAPSE || opMode == COMBINED)**

**{**

**do**

**{**

**ADMUX = (0 << MUX3) | (1 << MUX2) | (0 << MUX1) | (0 << MUX0)**

**| (0 << REFS0) | (0 << REFS1);**

**ADCSRA |= (1 << ADSC);**

**while( (ADCSRA&(1<<ADIF)) == 0 ); // wait for conversion**

**ADCV = ADC & 0x03FF; // grab only the 10 bits**

**voltage = (ADCV \* 5.0 / 0x3FF); // convert int to float**

**cam\_snap();**

**\_delay\_ms(60000);**

**}while(voltage > 2.5);**

**}**

**}**

**void ADC\_init()**

**{**

**DDRC = 0x0; // PORTC input**

**DIDR0 = 0x1; // disable digital input on ADC0 pin**

**// set reference to AREF**

**ADMUX = 0;**

**// initialize ADC**

**ADCSRA = 0x87; // enable ADC, system clock used for A/D conversion**

**ADCSRB = 0x0; // free running mode**

**TCCR1A |= (1 << COM1A1) | (1 << WGM11) | (1 << WGM10); //NON Inverted PWM**

**TCCR1B |= (0 << WGM13) | (1 << WGM12) | (1 << CS11) | (1 << CS10); //PRESCALER=64 MODE 14(FAST PWM)**

**if(reset)**

**{**

**ICR1 = 4999;**

**TCNT1 = 0;**

**}**

**TCCR3A |= (1 << COM3A1) | (1 << COM3B1) | (1 << WGM31) | (1 << WGM30); //NON Inverted PWM**

**TCCR3B |= (0 << WGM33) | (1 << WGM32) | (1 << CS31) | (1 << CS30); //PRESCALER=64 MODE 14(FAST PWM)**

**if(reset)**

**{**

**ICR3 = 4999;**

**TCNT3 = 0;**

**}**

**TCCR4A |= (1 << COM4A1) | (1 << COM4B1) | (1 << WGM41) | (1 << WGM40); //NON Inverted PWM**

**TCCR4B |= (0 << WGM43) | (1 << WGM42) | (1 << CS41) | (1 << CS40); //PRESCALER=64 MODE 14(FAST PWM)**

**if(reset)**

**{**

**ICR4 = 4999;**

**TCNT4 = 0;**

**}**

**DDRB |= (1 << PORTB0) | (1 << PORTB1);**

**DDRD |= (1 << PORTD0) | (1 << PORTD1) | (1 << PORTD2);**

**if(reset)**

**UART\_putStr("\n\rADC initialized. \n\r");**

**}**

**void cam\_init()**

**{**

**\_delay\_ms(100);**

**OCR1A = 166;**

**\_delay\_ms(100);**

**\_delay\_ms(100);**

**OCR3B = 127;**

**OCR4B = 127;**

**\_delay\_ms(100);**

**\_delay\_ms(100);**

**OCR1A = 366;**

**\_delay\_ms(100);**

**\_delay\_ms(100);**

**OCR3B = 366;**

**OCR4B = 366;**

**\_delay\_ms(100);**

**if(TCNT1 > 0)**

**UART\_putStr("Servos reset.\n\r");**

**if (SD.begin(chipSelect) && !SDcheck)**

**{**

**UART\_putStr("SD card initialized. \n\r");**

**SDcheck = TRUE;**

**}**

**else if(!SD.begin(chipSelect) && !SDcheck)**

**{**

**UART\_putStr("[ERROR] SD card initialization \n\r");**

**init\_fail();**

**return;**

**}**

**char \*reply = cam.getVersion();**

**if (!cam.begin() && reply == 0) {**

**UART\_putStr("[ERROR] Camera initialization \n\r");**

**init\_fail();**

**return;**

**}**

**else**

**{**

**cam.setImageSize(VC0706\_640x480);**

**UART\_putStr("Image size: 640x480 \n\r");**

**\_delay\_ms(10);**

**UART\_putStr("Camera initialized. \n\r");**

**init\_pass();**

**}**

**cam\_checkMode();**

**}**

**void cam\_checkMode()**

**{**

**ADMUX = (0 << MUX3) | (0 << MUX2) | (1 << MUX1) | (1 << MUX0)**

**| (0 << REFS0) | (0 << REFS1);**

**ADCSRA |= (1 << ADSC);**

**while( (ADCSRA&(1<<ADIF)) == 0 ); // wait for conversion**

**ADCV = ADC & 0x03FF; // grab only the 10 bits**

**voltage = (ADCV \* 5.0 / 0x3FF); // convert int to float**

**// dtostrf(voltage, 4, 3, voltageStr); // convert float to int**

**if(voltage > 3)**

**{**

**cam.setMotionDetect(true); // turn it on**

**opMode = MOTION;**

**if(cam.getMotionDetect())**

**if(reset)**

**UART\_putStr("Motion detection is ON. ");**

**else**

**if(reset)**

**UART\_putStr("[ERROR] Can't retrieve motion status.");**

**}**

**else if(voltage <= 3)**

**{**

**cam.setMotionDetect(false); // turn it off (default)**

**opMode = NORMAL;**

**if(reset)**

**UART\_putStr("Motion detection is OFF. ");**

**}**

**ADMUX = (0 << MUX3) | (1 << MUX2) | (0 << MUX1) | (0 << MUX0)**

**| (0 << REFS0) | (0 << REFS1);**

**ADCSRA |= (1 << ADSC);**

**while( (ADCSRA&(1<<ADIF)) == 0 ); // wait for conversion**

**ADCV = ADC & 0x03FF; // grab only the 10 bits**

**voltage = (ADCV \* 5.0 / 0x3FF); // convert int to float**

**// dtostrf(voltage, 4, 3, voltageStr); // convert float to int**

**if(voltage > 3)**

**{**

**if(reset)**

**UART\_putStr("Continuous mode ON. \n\r");**

**if(opMode == MOTION)**

**opMode = COMBINED;**

**else**

**opMode = LAPSE;**

**}**

**else if(voltage <= 3)**

**{**

**if(reset)**

**UART\_putStr("Continuous mode OFF. \n\r");**

**if(opMode == MOTION)**

**opMode = MOTION;**

**else**

**opMode = NORMAL;**

**}**

**}**

**void cam\_snap()**

**{**

**PORTD = (1 << PORTD4);**

**if (!cam.takePicture())**

**Serial.println("Failed to snap!");**

**else**

**Serial.println("Picture taken!");**

**char filename[13];**

**strcpy(filename, "IMAGE00.JPG");**

**for (int i = 0; i < 100; i++) {**

**filename[6] = '0' + i%10;**

**filename[5] = '0' + i/10;**

**// create if does not exist, do not open existing, write, sync after write**

**if (! SD.exists(filename)) {**

**break;**

**}**

**}**

**File imgFile = SD.open(filename, FILE\_WRITE);**

**uint16\_t jpglen = cam.frameLength();**

**Serial.print(jpglen, DEC);**

**Serial.println(" byte image");**

**Serial.print("Writing image to "); Serial.print(filename);**

**while (jpglen > 0) {**

**// read 32 bytes at a time;**

**uint8\_t \*buffer;**

**uint8\_t bytesToRead = min(64, jpglen); // change 32 to 64 for a speedup but may not work with all setups!**

**buffer = cam.readPicture(bytesToRead);**

**imgFile.write(buffer, bytesToRead);**

**jpglen -= bytesToRead;**

**}**

**imgFile.close();**

**Serial.println("...Done!");**

**cam.resumeVideo();**

**PORTD = (0 << PORTD4);**

**}**

**void init\_fail()**

**{**

**DDRD |= (1 << PORTD4);**

**PORTD |= (1 << PORTD4);**

**// init fail buzzer signal**

**PORTB = (1 << PORTB0);**

**\_delay\_ms(1000);**

**PORTB = (0 << PORTB0);**

**\_delay\_ms(40);**

**return;**

**}**

**void init\_pass()**

**{**

**UART\_putStr("\n\rInitialization process complete. \n\r\n\r");**

**DDRD |= (1 << PORTD4);**

**PORTD |= (1 << PORTD4);**

**// init complete buzzer signal**

**PORTB = (1 << PORTB0);**

**\_delay\_ms(40);**

**PORTB = (0 << PORTB0);**

**\_delay\_ms(40);**

**PORTB = (1 << PORTB0);**

**\_delay\_ms(40);**

**PORTB = (0 << PORTB0);**

**\_delay\_ms(40);**

**PORTB = (1 << PORTB0);**

**\_delay\_ms(40);**

**PORTB = (0 << PORTB0);**

**\_delay\_ms(40);**

**PORTD = (0 << PORTD4);**

**}**