**TITLE: PET DOOR SENSOR**

**GOAL:**

* Implement MPU6050 IMU sensor interfaced by I2C communication
* Use ESP8266 via serial communication to send sensor data to Ubidots
* Use Ubidots to receive data from the sensor and send an SMS/Email to user depending on the dog’s status
* Monitor whether the dog is inside or outside

**DELIVERABLES:**

This project is intended to implement an IMU along with cloud technology to notify users of the status of their dog. The use of cloud technology allows users to be remotely notified of whether the dog went outside or came back inside.

**LITERATURE SURVEY:**

There are around 80 million households in the USA that own dogs or cats. This means ~67% of all households in the USA have a cat or dog. Unfortunately, approximately 7.6 million dogs and cats enter animal shelters, which mean they have wandered out of their homes. This leads to a problem for pet owners, especially ones with traditional pet doors. These pet doors allow the pet with access to the outside and back home. However, most pet doors do not have a method of tracking the whereabouts of the animals. Even simple notifications indicating whether the pet is inside or outside the house do not exist. With the use of an IMU and Wi-Fi module, we can design such a product that will allow pet owners to worry less and monitor their pet’s whereabouts.

Using an IMU to detect angles about the X-axis will allow the pet door to signal different notifications depending on the angle. For example, if the IMU sends a value of -255 degrees, this can be signified as the “outside” notification that will be sent to the Wi-Fi module. The Wi-Fi module will send the data to a cloud site, such as Ubidots, which can send an SMS or Email if the value of the IMU is less than 0. The same can be said if the IMU sends a value of +255 degrees to the microcontroller. This will send a signal to send data to Ubidots and will send an SMS to the user notifying that the pet is inside the house. Using angles rather than motion thresholds allows multiple pets to be monitored. If two consecutive -255 values are received, this means two pets went outside; the same can be said about +255 values. The angles were calculated using the gyroscope and taking the integral of the gyroscopes rate to determine the angle of motion. This, however, can lead to issues in drifting results, where finding Euler angles using acceleration and a complementary filter can be used to correct drift.

The Wi-Fi module allows the user to send and receive data/notifications remotely. The AT+ firmware was used in this application to send commands to the module via USART interface. USART is used by sending commands serially to the module via the transmit port of the microcontroller. The microcontroller will then receive status signals from the Wi-Fi module to determine if the correct commands were issued and received correctly. The AT+ commands allows users to simply connect to the module without additional wires such as ISP or I2C interfaces.

The I2C/TWI interface is used to communicate with the MPU6050 IMU. This interface allows multiple devices to be connected via master-slave connections. The master is allowed to send data and request data and can communicate will all devices connected to the SDA/SCL line. Slaves can only communicate with the master. Communications occur when the SCL is high and stops when a STOP signal is received. Communications with multiple devices is achieved by addressing each device by their device address (usually hardwired, sometimes can be re-written). If multiple devices with the same address are present, the first device to respond to the master’s request will be communicated with. The device address consists of 7 bits, with the last bit representing either READ or WRITE operations. The I2C reads 9 bits, with the 9th bit being a STOP bit. Once a device is contacted, the master, depending on the 8th bit, can READ data from or WRITE to the slave.

Reads in I2C involves reading data with acknowledgement of received data. This bit is known as the acknowledged bit and is present on at the 9th bit position. Two versions of this bit exists, ACKNOWLEDGED and NOT ACKNOWLEDGED. If ACKNOWLEDGED is set at the 9th bit position, this tells the slave that the master will continue to read data. If NOT ACKNOWLEDGED is present, the slave will stop communication of data and return control to the master. Writes work in a similar fashion, writing and stopping once all bits are written to the register.

Communicating with a device after contacting is requires the program to point to the specific register to be read or written to, then starting the task after the starting register is chosen. If to be read/written are consecutive, a stop signal is not needed.

**COMPONENTS:**

**MPU6050 IMU**

The MPU6050 is an inertial motion unit (IMU) that has 6 degrees of freedom. Simply put, this means the device can measure 6 different motions: 3-axis gyroscope and 3-axis accelerometer. The device also has a temperature sensor, but is not used in this implementation. The MPU6050 is interfaced via I2C communication. With the use of the DMP (Digital Motion Processor) integrated on the MPU6050, users can also determine angles or implement methods of their own. The MPU6050 does not include a magnetometer like other IMUs, which limits its ability for more precise Euler angles calculations and determination of orientation.

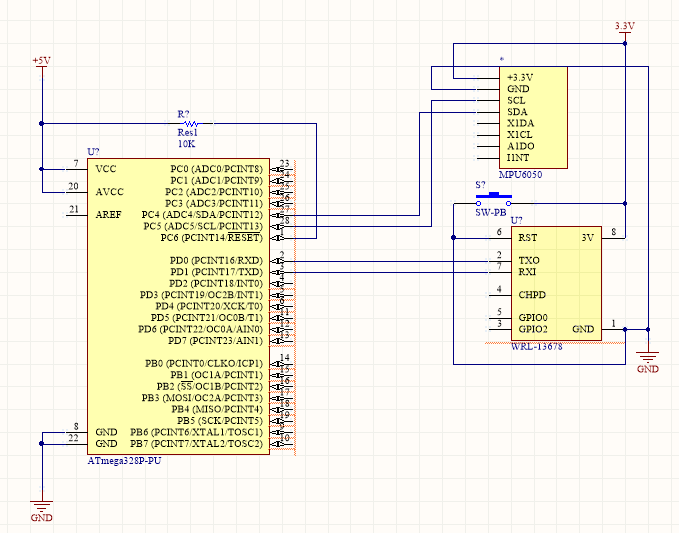
**ATmega328p**

The ATmega328p is an 8 bit microcontroller by Atmel. This microcontroller works by programming in either C or assembly. The code written will allow the users to determine what pins they want to set as input and output. This microcontroller consists of 32 general purpose registers. The ATmega328p consists of 6 PWMs, 3 timer/counters, up to 1kB EEPROM, 2kB SDRAM, I2C interface, SPI interface, 10-bit ADCs, and an 8MHz clock. The ATmega328p, however, is limited in the amount of pins can be used as input and output. Also, only one 16 bit timer exists on the chip and only one USART module is included.

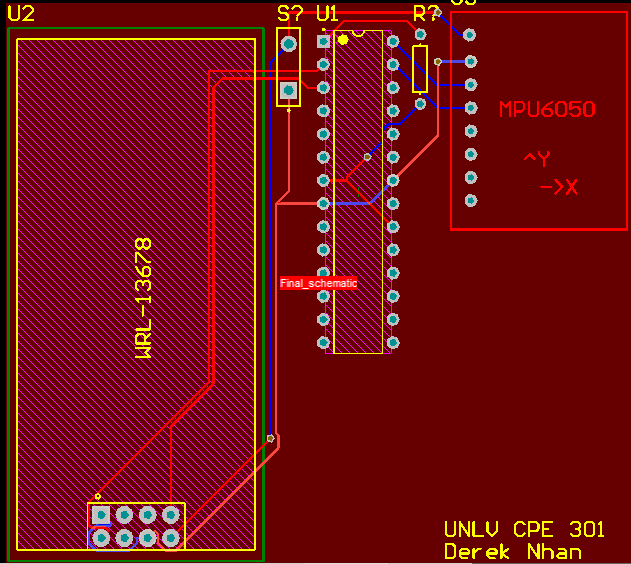
**ESP8266**

The ESP8266 is a Wi-Fi module that is capable of 2.4GHz communications. Different firmware can be flashed on the chip. In this application, we use AT+ firmware but is not the only method of communication. The AT+ firmware is used by sending AT+ commands to the ESP8266 via serial communication. The Wi-Fi module can be set up as a station or connect to an AP. Integrated TCP/IP protocols allow the module to communicate with cloud services to send and receive data. The ESP8266 is limited to only 1024 byte packets that can be received from a host. Working with AT+ commands, we are limited in only using serial communication via the TX/RX ports on the chip.

**SCHEMATICS:**



**INITIAL PCB\*:**



**IMPLEMENTATION:**

* MPU6050 IMU interfaced via I2C to receive commands from the microcontroller and to send data back to the microcontroller
  + The I2C communication is implemented via a header file adapted from g4lvanix via Github.
  + USART and the FTDI module was used to ensure correct data was being received from the IMU
* ESP8266 Wi-Fi module is used to receive from and send data serially to the microcontroller
  + The Wi-Fi module is interfaced through serial communications with the use of USART
  + FTDI USB-to-serial module was used to assure the correct baud rate, commands, and information received was used
  + Termite serial terminal was used to visualize data received from the Wi-Fi module
* ATmega328p was used to link the IMU and Wi-Fi module
  + The ATmega328p performed calculations to determine the angle of motion for the IMU
  + Sent trial information to the Wi-Fi module to ensure serial communications worked correctly

**SNAPSHOTS/SCREENSHOTS\*:** (only links - do not embed images or videos in the document)

Snapshots: <https://github.com/nhand2/CPE301S16>

MPU6050 Implementation Video: https://youtu.be/WMbKJY\_4CvA

IoT: https://youtu.be/6Cukp6D4Uw0

**CODE:**

**Main**

#define *F\_CPU* 8000000UL

#include <avr/io.h>

#include <util/twi.h>

#include <util/delay.h>

#include <stdlib.h>

#include <math.h>

#include "i2c\_control.h"

#include "MPU6050.h"

#include "usart.h"

#include "ESP8266.h"

char buffer[6]; //buffer to hold ASCII angle

int gyro\_xout = 0; //initialize gyro\_xout

int GYRO\_XOUT\_OFFSET = 0; //initialize GYRO offset

int GYRO\_XOUT\_OFFSET\_SUM = 0; //initialize sum of GYRO offset

int angle = 0; //initialize angle

int gyro\_rate = 0; //initialize the gyro\_rate

int gyro\_angle = 0; //initialize gyro\_angle

void get\_values(void)

{

i2c\_start(MPU6050\_WRITE); //set the address of MPU6050 write register

i2c\_write(GYRO\_YOUT\_H); //set pointer to GYRO\_XOUT\_H register

i2c\_stop();

i2c\_start(MPU6050\_READ); //set the address to READ register

gyro\_xout = ((*uint8\_t*) i2c\_read\_ack())<<8; //read the xout value and acknowledge data received (continue SLAVE to MASTER transmission)

gyro\_xout |= i2c\_read\_nack(); //read xout\_lower bits without acknowledge (stops SLAVE to MASTER transmission)

i2c\_stop(); //stop transmission

}

void calc\_values(void)

//This function calculates the angle of IMU. The angle is calculated by taking the gyro's x value and

//converting from degrees/second to degrees. The gyro contains a good amount of DRIFT from outside noise,

//the accelerometer is used to attempt to offset this noise

{

gyro\_rate = (gyro\_xout - GYRO\_XOUT\_OFFSET)/1.0131; //the gyro's rate is calculated

gyro\_angle += gyro\_rate/1000; //the angle is added to the rate divided by sample rate to get the angle of the IMU

*itoa*(gyro\_angle, buffer, 10);

}

void gyro\_calibrate(void)

//This function calibrates the GYROSCOPE by getting the value of the gyro's x-axis and adding the

//values. The values are then averaged and the OFFSET of the gyroscope is obtained

{

for (int i=0; i<1000; i++) //loop for 1000 iterations

{

get\_values(); //get the GYRO values by calling get\_values

GYRO\_XOUT\_OFFSET\_SUM += gyro\_xout; //add next gyro\_xout value to offset sum

}

GYRO\_XOUT\_OFFSET = GYRO\_XOUT\_OFFSET\_SUM/1000; //average the OFFSET to get the official offset

}

void MPU6050\_init(void)

//This function initialize the MPU6050 by sending various commands to the IMU

{

i2c\_start(MPU6050\_WRITE); //sets WRITE address of device

i2c\_write(PWR\_MGMT\_1); //sets address of power management 1

i2c\_write(0x00); //turns on the device

i2c\_stop(); //stop I2C connection

i2c\_start(MPU6050\_WRITE);

i2c\_write(CONFIG); //set address to point to CONFIG register

i2c\_write(0x01); //write 0x01 to the register ()

i2c\_stop();

i2c\_start(MPU6050\_WRITE);

i2c\_write(GYRO\_CONFIG); //set address to point to GYRO\_CONFIG register

i2c\_write(0x10); //write 0x10 to register (1000 degrees/sec range)

i2c\_stop();

i2c\_start(MPU6050\_WRITE);

i2c\_write(ACCEL\_CONFIG); //configures the accelerometer

i2c\_write(0x00); //write 0x00 to register (+/- 2g range)

i2c\_stop();

i2c\_start(MPU6050\_WRITE);

i2c\_write(INT\_ENABLE); //set pointer to address of INTERRUPTS ENABLE

i2c\_write(0x00); //write 0x00 (no interrupts)

i2c\_stop();

i2c\_start(MPU6050\_WRITE);

i2c\_write(SIGNAL\_PATH\_RESET); //set pointer to address of

i2c\_write(0x00); //write 0x00 (no reset)

i2c\_stop();

}

int main()

{

PINC = 0xFF; //set pull up resistors to ensure proper data

i2c\_init(); //initialize all I2C/TWI interaface registers/module

usart\_init(); //initialize usart module on ATmega328p

MPU6050\_init(); //call to initialize the MPU6050

gyro\_calibrate(); //call to calibrate the gyro to zero it at a given position

char \*SSID = "Mah\_New\_Internet"; //AP SSID

*uint8\_t* connected = 0; //variable to check if connected

char \*HTTP\_POST; //character array/string to hold the generic "HTTP POST" command

char cmd[256]; //buffer to hold the "POST" command, variable depending on the size and variable to be sent

char cipsend[32]; //buffer to hold the cipsend command, which is variable according to # of bytes sending

*uint8\_t* err; //variable to hold error status

while (1)

{

get\_values(); //get the values from the gyro and accel

calc\_values(); //call function to calculate the values of the accel and gyro

if (gyro\_angle > 200|| gyro\_angle < -400)

{

usart\_sends("send");

ESP8266\_cmd("AT+RST\r\n", 1); //AT+ command to reset the ESP8266

while (ESP8266\_rec("WIFI GOT IP", *NULL*)); //wait here until WIFI is initialized correctly

ESP8266\_cmd("ATE0\r\n", 1); //disable ECHO

ESP8266\_cmd("AT+CWJAP?\r\n", 0); //check connected AP

if (ESP8266\_rec(SSID, *NULL*) == 0) //checks if AP is correct and set "conencted" to 1

connected = 1;

if (connected) //enter if connected to AP

{

ESP8266\_cmd("AT+CIPMODE=0\r\n", 1); //set CIPMODE to 0

ESP8266\_cmd("AT+CIPMUX=0\r\n", 1); //set CIPMUX to 0, single port connection

ESP8266\_cmd("AT+CIPSTART=\"TCP\",\"50.23.124.68\",80\r\n", 0); //connect to Ubidots server using CIPSTART

err = ESP8266\_rec("CONNECT", *NULL*); //store results of command to err variable

if(!err) //if the connected, enter

{

//the following line sets the POST command to the ESP8266 to SEND data to the server

HTTP\_POST="POST /api/v1.6/variables/572c0ff97625420e3376b14c/values HTTP/1.1\r\nX-Auth-Token: QUAq0Xx3zPAPxpxDb6JWzL8v62TJtc\r\nHost: things.ubidots.com\r\nContent-Type: application/json\r\nContent-Length: %d\r\n\r\n{\"value\":%s}";

*snprintf*(cmd, sizeof(cmd),HTTP\_POST, *strlen*(buffer)+10, buffer); //prints the string in to a string that is variable in size but cannot exceed max sizeof(cmd)

cmd[sizeof(cmd)-1] = '\0'; //null terminate the string

*snprintf*(cipsend, sizeof(cipsend), "AT+CIPSEND=%d\r\n", *strlen*(cmd)); //prints the string with variable byte size

cipsend[sizeof(cipsend)-1] = '\0'; //null terminate the string

ESP8266\_cmd(cipsend, 0); //call the send command of ESP8266

ESP8266\_rec(">", *NULL*); //check to see if '>' is recieved

ESP8266\_cmd(cmd, 0); //send the command printed to the cmd buffer

err = ESP8266\_rec("SEND OK", *NULL*); //check if "SEND OK" was issued from the ESP8266

ESP8266\_cmd("AT+CIPCLOSE\r\n", 0); //terminate the connection once done

ESP8266\_rec("OK", "ERROR"); //check the result of closing the serve connection

}

gyro\_calibrate();

gyro\_angle = 0;

}

}

}

return 0;

}

**ESP8266**

#include <avr/io.h>

#include "usart.h"

#include <string.h>

#include <stdlib.h>

static char reply[64];

static *uint8\_t* reply\_pos; //pointer for reply from ESP8266

*uint8\_t* ESP8266\_rec(const char\* k\_reply, const char\* n\_reply)

//This function works to receive a reply from the the ESP8266

//Two strings are accepted, okay and not\_okay. Okay will send 0, meaning success

//Not okay will send 1, which means failed

{

reply\_pos = 0; //position of the reply

for (;;) //similiar to while loop

{

*uint16\_t* rx = usart\_receive(); //save receive in rx

if ((rx>>8) == 0) //if rx shifted 8 bits right == 0 do this

{

reply[reply\_pos++] = rx & 0xFF; //determine what character is in rx

reply[reply\_pos] = '\0'; //NULL terminator

if (reply\_pos == sizeof(reply)-1) //reset position if end of string

{

reply\_pos=0;

}

if (k\_reply && *strstr*(reply, k\_reply)) //if okay matched, return 0

return 0;

if (n\_reply && *strstr*(reply, n\_reply)) //if not okay matched, return 1

{

return 1;

}

}

}

}

*uint8\_t* ESP8266\_cmd (char\* cmd, *uint8\_t* wait\_ok)

//This function takes in a command for the ESP8266 and will wait until an "OK" is received

{

usart\_sends(cmd); //send the command via USART

if(wait\_ok) //if wait\_ok is 1, we expect and "OK" reply

{

return ESP8266\_rec("OK\r\n", *NULL*); //return if OK was received

}

return 0;

}

**I2C**

#define *F\_CPU* 8000000UL

#include <avr/io.h>

#include <util/twi.h>

#include "i2c\_control.h"

#define F\_SCL 10000UL //CLOCK for TWI

#define PRESCALAR 1 //prescalar

#define TWBAUD (((*F\_CPU*/F\_SCL)-16)/2) //set the baud rate of the TWI

void i2c\_init(void)

//This function initialize the TWI interface

{

TWSR = 0; //set status register to 0

TWBR = (*uint8\_t*) TWBAUD; //set TWBR with calculated baud rate

}

*uint8\_t* i2c\_start(*uint8\_t* addr)

{

*uint8\_t* status; //variable to hold status

//TWCR = 0; //resets the control register

//set TWI interrupts, set start bit, enable 2-wire

TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

//wait for end of trasmit

//2-wire interrupt = 0

while (!(TWCR & (1<<TWINT)));

if ((TWSR & 0xF8) != *TW\_START*)

//if TWSR is not the same as start, error

return 1;

//load slave address

TWDR = addr;

//start trasmit

TWCR = (1<<TWINT) | (1<<TWEN);

//wait for trasmit to finish

while (!(TWCR & (1<<TWINT)));

//check for acknowledge bit

status = *TW\_STATUS* & 0xF8;

if ((status != *TW\_MT\_SLA\_ACK*) && (status != *TW\_MR\_SLA\_ACK*))

//if status is not acknowledge bit, send an error

return 1;

return 0;

}

*uint8\_t* i2c\_write(*uint8\_t* data)

{

TWDR = data; //load data into TWDR

TWCR = (1<<TWINT) | (1<<TWEN); //start transmit

while (!(TWCR & (1<<TWINT))); //wait for transmit to finish

if ((TWSR & 0xF8) != *TW\_MT\_DATA\_ACK*)

//if master transmit is not acknowledge, send error

return 1;

return 0;

}

*uint8\_t* i2c\_read\_ack(void)

{

//start 2-wire and acknowledge data after finished

TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);

//wait for transmit to finish

while (!(TWCR & (1<<TWINT)));

return TWDR; //return data received

}

*uint8\_t* i2c\_read\_nack(void)

{

//start receive w/o acknowledge bit

TWCR = (1<<TWINT) | (1<<TWEN);

//wait for transmit end

while (!(TWCR & (1<<TWINT)));

//return received data

return TWDR;

}

void i2c\_stop(void)

{

TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO); //stop trasmit

while(TWCR & (1<<TWSTO));

}

**USART**

#define *F\_CPU* 8000000UL

#include <avr/io.h>

#include <stdlib.h>

#include "usart.h"

void usart\_init(void)

//This function intializes the USART module on the ATmega328p

{

UCSR0B |= (1<<TXEN0)|(1<<RXEN0); //enable transmit only

UCSR0C |= ((1<<USBS0)|(3<<UCSZ00)); //removed (1<<UMSEL00) because it sets in synchronous mode, we are using asynchronous

UBRR0H = 0x00; //high value of baud rate

UBRR0L = 0x19; //baud rate of 19200

}

void usart\_send(unsigned char data)

//This function takes a character datatype and loads it in to the register UDR0 for tramission

{

while (!(UCSR0A & (1<<UDRE0))); //wait here until buffer is empty

UDR0 = data; //put DATA into buffer

}

void usart\_sends(char \*s)

//This function takes in a string/character array and sends it one byte at a time to transmit a string

{

while(\*s) //while there are still characters in the string

{

usart\_send(\*s); //call usart\_send to send a single byte

s++; //advance the string position

}

}

unsigned int usart\_receive(void)

//This function receives data from the RX0 pin and assigns that byte to the data buffer

{

unsigned char data=0; //clear data of previous information

while (!(UCSR0A & (1<<RXC0))); //wait until receive is complete

data=UDR0; //assign data to UDR0

return data; //return the byte of data

}

**REFERENCE:**

**ATmega328p Datasheet:** <http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet_complete.pdf>

**ESP8266 Datasheet:** <https://cdn-shop.adafruit.com/datasheets/ESP8266_Specifications_English.pdf>

**ESP8266 Libraries:** <https://tinyurl.com/unlvcpe301s16>

**Gyroscope Angles:** <http://www.pieter-jan.com/node/7>

**I2C Library:** <https://github.com/g4lvanix/I2C-master-lib>

**MPU6050 Datasheet:** <https://www.cdiweb.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf>

**Pet Statistics:** <http://www.aspca.org/animal-homelessness/shelter-intake-and-surrender/pet-statistics>

<https://www.avma.org/KB/Resources/Statistics/Pages/Market-research-statistics-US-pet-ownership.aspx>

**USART Libraries:** <http://jump.to/fleury>