Appendix C – Server code

```
1. #include <netinet/in.h>
2. #include <unistd.h>
3. #include <stdio.h>
4. #include <stdlib.h>
5. #include <arpa/inet.h>
6. #include <string.h>
7. #include <signal.h>
8. #include <stdio_ext.h>
9. #include <sys/wait.h>
10. #include <errno.h>
11. int n_chld;
12. typedef struct {
13. char line[65];
14. }micro_info;
15. void sigchld_handler(int signum)
16. {
17. int pid;
18. while((pid=waitpid(-1,NULL,WNOHANG))>0)
19. {
20. n_chld--;
21. printf("proc %d ended",pid);
22. }
23. }
```

```
24. int create_socket(int port);
25. void atend_pedido(int);
26. int main(int argc, char *argv[]){
27. int ns, s;
28. unsigned int clilen;
29. struct sockaddr_in cli_addr;
30. n_chld=0;
31. if(argc!=2) {
32. printf("Usage: %s port_number\n",argv[0]);
33. exit(1);
34. }
35. s = create_socket(atoi(argv[1]));
36. while(1) {
37. printf("Waiting connection\n");
38. clilen = sizeof(cli_addr);
39. ns = accept(s, (struct sockaddr *) &cli_addr, &clilen);
40. if((ns<0) && (errno==EINTR))
41. continue;
42. printf("Connection from ip %s, port %d\n", inet_ntoa(*((struct in_addr *)
    &(cli_addr.sin_addr))), ntohs(cli_addr.sin_port));
43. atend_pedido(ns);
44. }
45. return 0;
46. }
47. void atend_pedido(int ns)
48. {
```

```
49. int r;
50. FILE *fp=fdopen(ns,"r+");
51. FILE *ptr;
52. struct sigaction act;
53. act.sa_handler=sigchld_handler;
54. act.sa_flags=SA_NOCLDSTOP;
55. sigemptyset(&(act.sa_mask));
56. sigaction(SIGCHLD, &act, NULL);
57. sigset_t mask;
58. sigemptyset(&mask);
59. sigaddset(&mask, SIGCHLD);
60. char aux[65];
61. micro_info v;
62. if(fp==NULL) {
63. perror("fdopen");
64. close(ns);
65. return;
66. }
67. char op = fgetc(fp);
68. ptr = fopen("text.txt", "w");
69. fread(v.line,65,1,fp);
70. signal(SIGCHLD, SIG_DFL);
71. while(n_chld>0)
72. {
73. printf("",n_chld)
74. wait(NULL);
```

```
75. n_chld--;
76. }
77. sigprocmask(SIG_BLOCK,&mask,NULL);
78. fwrite(&v,sizeof(micro_info),1,ptr);
79. sigprocmask(SIG_UNBLOCK,&mask,NULL);
80. sleep(2);
81. printf("Line: %.100s\n",v.line);
82. sigaction(SIGCHLD, &act, NULL);
83. fclose(ptr);
84. fclose(fp);
85. }
86. int create_socket(int port)
87. {
88. int s;
89. struct sockaddr_in serv_addr;
90. s = socket(PF_INET, SOCK_STREAM, 0);
91. if(s<0)
92. {
93. perror("socket");
94. exit(1);
95. }
96. memset(&serv_addr, 0, sizeof(serv_addr));
97. serv_addr.sin_family = AF_INET;
98. serv_addr.sin_addr.s_addr = INADDR_ANY;
99. serv_addr.sin_port = htons(port);
100.         if(bind(s, (struct sockaddr *)&serv_addr, sizeof(serv_addr))<0)
```

```
101.        {
102.        perror("bind");
103.        exit(1);
104.        }
105.        listen(s,5);
106.        return(s);
107.        }
```

Appendix D – Arduino code

```
/*
Credits to

Nathan Seidle (SparkFun Electronics) for the base code "MMA8452Q Basic Example Code" - Modified to fit our needs (data rates and outputs)
Daniel Gonçalves a.k.a. (t.c.p.) Tr3s for the base code of the EEPROM read/write routines - Modified to work with the 24LC1025 I2C EEPROM
Creators of the TinyGPS library for the example code - Modified to fit our needs

*/

#include <SoftwareSerial.h>
#include <TinyGPS.h>
#include <Wire.h>
#include <PString.h>

#define eeprom 0x50            // EEPROM address

#define MMA8452_ADDRESS 0x1D
#define OUT_X_MSB 0x01
#define XYZ_DATA_CFG  0x0E
#define WHO_AM_I   0x0D
#define CTRL_REG1  0x2A
#define GSCALE 4 // Sets full-scale range to +/-2, 4, or 8g. Used to calc real g values.

#define EEPROM_VCC 10
#define ACCELEROMETER_VCC 11
```

```
#define GPS_VCC 12
#define ONOFFKEY 7

TinyGPS gps;
SoftwareSerial nss(3, 4);
SoftwareSerial cell(8,9);

static void gpsdump(TinyGPS &gps);
static bool feedgps();
static void print_float(float val, float invalid, int len, int prec);
static void print_int(unsigned long val, unsigned long invalid, int len);
static void print_date(TinyGPS &gps);
static void print_str(const char *str, int len);

char buffer[65];
unsigned short int buff_i=0;

static const float SAFE_LAT1 = 41.17946, SAFE_LAT2 = 41.17925, SAFE_LON1 = -8.60569, SAFE_LON2 = -8.60949;
float x, y, z, lat, lon;
float activity[6];
const byte dataRate=7;     //0=800 Hz, 1=400 Hz, 2=200 Hz, 3=100 Hz, 4=50 Hz, 5=12.5 Hz, 6=6.25 Hz, 7=1.56 Hz
unsigned short int k, operatingmode, i, j;
char gps_time[6];

PString myString(buffer,sizeof(buffer));

void setup()
{
  Serial.begin(115200);
  nss.begin(4800);
  Wire.begin();

```

```
  digitalWrite(EEPROM_VCC, LOW);
  digitalWrite(ONOFFKEY, LOW);
  digitalWrite(ACCELEROMETER_VCC, LOW);
  digitalWrite(GPS_VCC, HIGH);
}

void loop()
{
  while( get_location() == false )
  {
    setup_GSM();
    tracking_mode();
  }
  digitalWrite(GPS_VCC, LOW);
  digitalWrite(ONOFFKEY, LOW);
  digitalWrite(ACCELEROMETER_VCC, HIGH);
  activity_monitoring();
  digitalWrite(ACCELEROMETER_VCC, LOW);
  digitalWrite(EEPROM_VCC, HIGH);
  send_data();
  digitalWrite(EEPROM_VCC, LOW);
  digitalWrite(GPS_VCC, HIGH); }

/******************************  TRACKING  MODE
  *******************************/

void tracking_mode()
{
  while(safe_zone() == false)
  {
  send_data();
  get_location();
  }
}
```

```
91.
92. /******************************                    ACCELEROMETER
    ******************************/
93.
94. /*
95.  MMA8452Q Basic Example Code
96.  Nathan Seidle
97.  SparkFun Electronics
98.  November 5, 2012
99.
100.
101.         Hardware setup:
102.
103.         MMA8452---------------------- Arduino
104.
105.         3.3V ------------------------- 3.3 V
106.         SDA -------------------------- A4
107.         SCL -------------------------- A5
108.         GND -------------------------- GND
109.
110.         */
111.
112.        void activity_monitoring()
113.        {
114.          initMMA8452();
115.          delay(1000);
116.          int accelCount[3];  // Stores the 12-bit signed value
117.          readAccelData(accelCount);  // Read the x/y/z adc values
118.          // Now we'll calculate the accleration value into actual g's
119.          float accelG[3];  // Stores the real accel value in g's
120.          for (i = 0 ; i < 3 ; i++)
121.          {
122.            accelG[i] = (float) accelCount[i] / ((1<<12)/(2*GSCALE));  // get actual
    g value, this depends on scale being set
```

```
123.          }
124.
125.          x+=accelG[0];
126.          y+=accelG[1];
127.          z+=accelG[2];
128.
129.          k++;
130.          if( k >= 936 )      // Calculates the average after 10 minutes of reading data
131.          {
132.            j++;
133.            activity[j]=x+y+z/(3*k);
134.            x=0;y=0;z=0;k=0;
135.            if( j == 6 )
136.              return;
137.          }
138.        }
139.
140.        void readAccelData(int *destination)
141.        {
142.          byte rawData[6];  // x/y/z accel register data stored here
143.          readRegisters(OUT_X_MSB, 6, rawData);   // Read the six raw data
    registers into data array
144.          // Loop to calculate 12-bit ADC and g value for each axis
145.          for(int i = 0; i < 3 ; i++)
146.          {
147.            int gCount = (rawData[i*2] << 8) | rawData[(i*2)+1];  //Combine the two
    8 bit registers into one 12-bit number
148.            gCount >>= 4; //The registers are left align, here we right align the 12-bit
    integer
149.
150.            // If the number is negative, we have to make it so manually (no 12-bit
    data type)
151.            if (rawData[i*2] > 0x7F)
152.            {
```

```
153.              gCount = ~gCount + 1;
154.              gCount *= -1;  // Transform into negative 2's complement #
155.            }
156.            destination[i] = gCount; //Record this gCount into the 3 int array
157.          }
158.        }
159.
160.
161.        void initMMA8452()
162.        {
163.          // Setup the 3 data rate bits, from 0 to 7
164.          writeRegister(0x2A, readRegister(0x2A) & ~(0x38));
165.          if (dataRate <= 7)
166.            writeRegister(0x2A, readRegister(0x2A) | (dataRate << 3));
167.
168.          byte c = readRegister(WHO_AM_I);  // Read WHO_AM_I register
169.          if (c == 0x2A) // WHO_AM_I should always be 0x2A  //debug
170.          {
171.            Serial.println("MMA8452Q is online...");
172.          }
173.          else
174.          {
175.            Serial.print("Could not connect to MMA8452Q: 0x");
176.            Serial.println(c, HEX);
177.            while(1) ; // Loop forever if communication doesn't happen
178.          }
179.
180.          MMA8452Standby();  // Must be in standby to change registers
181.
182.          // Set up the full scale range to 2, 4, or 8g.
183.          byte fsr = GSCALE;
184.          if(fsr > 8) fsr = 8; //Easy error check
185.          fsr >>= 2;
186.          writeRegister(XYZ_DATA_CFG, fsr);
```

```
187.
188.          MMA8452Active();  // Set to active to start reading
189.        }
190.
191.        // Sets the MMA8452 to standby mode. It must be in standby to change most register settings
192.        void MMA8452Standby()
193.        {
194.          byte c = readRegister(CTRL_REG1);
195.          writeRegister(CTRL_REG1, c & ~(0x01)); //Clear the active bit to go into standby
196.        }
197.
198.        // Sets the MMA8452 to active mode. Needs to be in this mode to output data
199.        void MMA8452Active()
200.        {
201.          byte c = readRegister(CTRL_REG1);
202.          writeRegister(CTRL_REG1, c | 0x01); //Set the active bit to begin detection
203.        }
204.
205.        // Read bytesToRead sequentially, starting at addressToRead into the dest byte array
206.        void readRegisters(byte addressToRead, int bytesToRead, byte * dest)
207.        {
208.          Wire.beginTransmission(MMA8452_ADDRESS);
209.          Wire.write(addressToRead);
210.          Wire.endTransmission(false); //endTransmission but keep the connection active
211.
212.          Wire.requestFrom(MMA8452_ADDRESS, bytesToRead); //Ask for bytes, once done, bus is released by default
213.
```

```
214.          while(Wire.available() < bytesToRead); //Hang out until we get the # of
    bytes we expect
215.
216.          for(int x = 0 ; x < bytesToRead ; x++)
217.            dest[x] = Wire.read();
218.        }
219.
220.        // Read a single byte from addressToRead and return it as a byte
221.        byte readRegister(byte addressToRead)
222.        {
223.          Wire.beginTransmission(MMA8452_ADDRESS);
224.          Wire.write(addressToRead);
225.          Wire.endTransmission(false); //endTransmission but keep the connection
    active
226.
227.          Wire.requestFrom(MMA8452_ADDRESS, 1); //Ask for 1 byte, once done,
    bus is released by default
228.
229.          while(!Wire.available()) ; //Wait for the data to come back
230.          return Wire.read(); //Return this one byte
231.        }
232.
233.        // Writes a single byte (dataToWrite) into addressToWrite
234.        void writeRegister(byte addressToWrite, byte dataToWrite)
235.        {
236.          Wire.beginTransmission(MMA8452_ADDRESS);
237.          Wire.write(addressToWrite);
238.          Wire.write(dataToWrite);
239.          Wire.endTransmission(); //Stop transmitting
240.        }
241.
242.        /************************************ GSM
    ************************************/
243.        /*
```

```
244.          Hardware setup:
245.
246.          ADH8066---------------------- Arduino
247.
248.          VBAT -------------------------- 3.3 V
249.          SDA --------------------------- A4
250.          SCL --------------------------- A5
251.          GND --------------------------- GND
252.
253.         */
254.         void setup_GSM()
255.         {
256.           delay(30000);
257.           cell.println("ATE0");    //switch off echo
258.           cell.println("+AIPDCONT=\"myconnection\",\"\",\"\"");          //connect to
    internet (Optimus) [APN, username, password]
259.           cell.println("AT+AIPA=1");
260.           cell.println("AT+AIPO=1,,\"xxx.xxx.xxx.xxx\",1500,0");     //Open socket
    ID 1, connect to server at IP "xxx.xxx.xxx.xxx", Port number 1500, tcp mode
261.           cell.flush();
262.         }
263.
264.         void send_data()
265.         {
266.           setup_GSM();
267.           cell.print("AT+AIPW=1,");
268.           cell.print(myString);    //prints buffer useing ASCII charracters
269.           cell.println("\"");
270.           cell.flush();
271.           // Close socket and connection
272.           cell.println("AT+AIPC=1");
273.           cell.println("AT+AIPA=0");
274.           cell.flush();
275.         }
```

```
276.
277.
278.        /*********************************                    EEPROM
    **********************************/
279.
280.         /*Hardware setup:
281.
282.         24LC1025---------------------- Arduino
283.
284.         3.3V ------------------------- 3.3 V
285.         SDA -------------------------- A4
286.         SCL -------------------------- A5
287.         GND -------------------------- GND
288.
289.        */
290.        void buffer_write(void)
291.        {
292.          myString.print(operatingmode, DEC);
293.          myString.print(gps_time);
294.          myString.print(lat, DEC);
295.          myString.print(lon, DEC);
296.          myString.print(activity[0],DEC);
297.          myString.print(activity[1],DEC);
298.          myString.print(activity[2],DEC);
299.          myString.print(activity[3],DEC);
300.          myString.print(activity[4],DEC);
301.          myString.print(activity[5],DEC);
302.
303.          for(unsigned short int b = 0; b<56; b++)
304.          {
305.            writeEEPROM(eeprom, b, buffer[b]);
306.            if ( b==0 || b==6 || b==14 || b==22 || b==29 || b==36 || b==43 || b==50 ||
    b==57 )    //writes the information to EEPROM in the form [Operating mode, time,
    lat, lon, activity_1, activity_2, activity_3, activity_4, activity_5, activity_6]
```

```
307.              writeEEPROM(eeprom,                b,                ',');
    // 0 - Regular
308.           }
    // 1 - Tracking
309.           myString.begin();  //clear string
310.         }
311.
312.         void buffer_read(void)
313.         {
314.           unsigned short int b;
315.           for(b = 0; b<65; b++)
316.           {
317.             buffer[b]=readEEPROM(eeprom, b);
318.           }
319.         }
320.
321.         void writeEEPROM(int deviceaddress, unsigned int eeaddress, byte data )
322.         {
323.           Wire.beginTransmission(deviceaddress);
324.           Wire.write((int)(eeaddress >> 8));   // MSB
325.           Wire.write((int)(eeaddress & 0x400)); // LSB
326.           Wire.write(data);
327.           Wire.endTransmission();
328.         }
329.
330.         byte readEEPROM(int deviceaddress, unsigned int eeaddress )
331.         {
332.           byte rdata = 0x400;
333.           Wire.beginTransmission(deviceaddress);
334.           Wire.write((int)(eeaddress >> 8));   // MSB
335.           Wire.write((int)(eeaddress & 0x400)); // LSB
336.           Wire.endTransmission();
337.           Wire.requestFrom(deviceaddress,1);
338.           if (Wire.available())
```

```
339.            rdata = Wire.read();
340.           return rdata;
341.         }
342.
343.         /************************************ GPS
    ************************************/
344.
345.         /*
346.           Hardware setup:
347.
348.             GPS ----- Arduino
349.
350.             GND--------GND
351.             VCC--------D12
352.             Enable-----D12
353.             Tx---------D3
354.             Rx---------D4
355.         */
356.
357.         boolean safe_zone()
358.         {
359.           if( lat < SAFE_LAT1 && lat > SAFE_LAT2 && lon < SAFE_LON1 &&
    lon > SAFE_LON2 )
360.             return true;
361.           else
362.             return false;
363.         }
364.
365.         boolean get_location()
366.         {
367.           bool newdata = false;
368.           unsigned long start = millis();
369.
370.           // Every second we print an update
```

```
371.          while (millis() - start < 1000)
372.          {
373.            if (feedgps())
374.              newdata = true;
375.          }
376.
377.          gpsdump(gps);
378.         if( lat < SAFE_LAT1 && lat > SAFE_LAT2 && lon < SAFE_LON1 &&
    lon > SAFE_LON2 )
379.            return true;
380.          else
381.            return false; }
382.
383.        static void gpsdump(TinyGPS &gps)
384.        {
385.          float flat, flon;
386.          unsigned long age, date, time, chars = 0;
387.          unsigned short sentences = 0, failed = 0;
388.
389.          print_int(gps.satellites(), TinyGPS::GPS_INVALID_SATELLITES, 5);
390.          print_int(gps.hdop(), TinyGPS::GPS_INVALID_HDOP, 5);
391.          gps.f_get_position(&flat, &flon, &age);
392.          lat=flat;
393.          lon=flon;
394.          print_float(flat, TinyGPS::GPS_INVALID_F_ANGLE, 9, 5);
395.          print_float(flon, TinyGPS::GPS_INVALID_F_ANGLE, 10, 5);
396.          print_int(age, TinyGPS::GPS_INVALID_AGE, 5);
397.
398.          print_date(gps);
399.
400.          print_float(gps.f_altitude(),  TinyGPS::GPS_INVALID_F_ALTITUDE,  8,
    2);
401.          print_float(gps.f_course(), TinyGPS::GPS_INVALID_F_ANGLE, 7, 2);
```

```
402.          print_float(gps.f_speed_kmph(), TinyGPS::GPS_INVALID_F_SPEED, 6,
    2);
403.          print_str(gps.f_course() == TinyGPS::GPS_INVALID_F_ANGLE ? "***
    " : TinyGPS::cardinal(gps.f_course()), 6);
404.          print_int(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0UL : (unsigned
    long)TinyGPS::distance_between(flat, flon, SAFE_LAT1, SAFE_LON1) / 1000,
    0xFFFFFFFF, 9);
405.          print_float(flat == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 :
    TinyGPS::course_to(flat, flon, 51.508131, -0.128002),
    TinyGPS::GPS_INVALID_F_ANGLE, 7, 2);
406.          print_str(flat == TinyGPS::GPS_INVALID_F_ANGLE ? "*** " :
    TinyGPS::cardinal(TinyGPS::course_to(flat, flon, SAFE_LAT1, SAFE_LON1)),
    6);
407.
408.          gps.stats(&chars, &sentences, &failed);
409.          print_int(chars, 0xFFFFFFFF, 6);
410.          print_int(sentences, 0xFFFFFFFF, 10);
411.          print_int(failed, 0xFFFFFFFF, 9);
412.          Serial.println();
413.        }
414.
415.        static void print_int(unsigned long val, unsigned long invalid, int len)
416.        {
417.          char sz[32];
418.          if (val == invalid)
419.            strcpy(sz, "*******");
420.          else
421.            sprintf(sz, "%ld", val);
422.          sz[len] = 0;
423.          for (int i=strlen(sz); i<len; ++i)
424.            sz[i] = ' ';
425.          if (len > 0)
426.            sz[len-1] = ' ';
427.          Serial.print(sz);
```

```
428.          feedgps();
429.        }
430.
431.        static void print_float(float val, float invalid, int len, int prec)
432.        {
433.          char sz[32];
434.          if (val == invalid)
435.          {
436.            strcpy(sz, "*******");
437.            sz[len] = 0;
438.                if (len > 0)
439.                  sz[len-1] = ' ';
440.            for (int i=7; i<len; ++i)
441.                sz[i] = ' ';
442.            Serial.print(sz);
443.          }
444.          else
445.          {
446.            Serial.print(val, prec);
447.            int vi = abs((int)val);
448.            int flen = prec + (val < 0.0 ? 2 : 1);
449.            flen += vi >= 1000 ? 4 : vi >= 100 ? 3 : vi >= 10 ? 2 : 1;
450.            for (int i=flen; i<len; ++i)
451.              Serial.print(" ");
452.          }
453.          feedgps();
454.        }
455.
456.        static void print_date(TinyGPS &gps)
457.        {
458.          int year;
459.          byte month, day, hour, minute, second, hundredths;
460.          unsigned long age;
```

```
461.          gps.crack_datetime(&year, &month, &day, &hour, &minute, &second,
    &hundredths, &age);
462.          if (age == TinyGPS::GPS_INVALID_AGE)
463.            Serial.print("*******    *******    ");
464.          else
465.          {
466.            char sz[32];
467.            sprintf(sz, "%02d/%02d/%02d %02d:%02d:%02d   ",
468.                month, day, year, hour, minute, second);
469.            sprintf(gps_time, "%02d:%02d", hour, minute);
470.
471.            Serial.print(sz);
472.          }
473.          print_int(age, TinyGPS::GPS_INVALID_AGE, 5);
474.          feedgps();
475.        }
476.
477.        static void print_str(const char *str, int len)
478.        {
479.          int slen = strlen(str);
480.          for (int i=0; i<len; ++i)
481.            Serial.print(i<slen ? str[i] : ' ');
482.          feedgps();
483.        }
484.
485.        static bool feedgps()
486.        {
487.          while (nss.available())
488.          {
489.            if (gps.encode(nss.read()))
490.              return true;
491.          }
492.          return false;
493.        }
```