Smart Office: Wireless Sensor Network for Energy Monitoring and User Profiling

ORIOL PRATS VIDAL

KTH Electrical Engineering

Master's Degree Project
Stockholm, Sweden September 2010

XR-EE-RT 2010:010

Kungliga Tekniska Högskolan

Automatic Control Lab

Smart Office: Wireless Sensor Network for Energy Monitoring and User Profiling

Electrical Engineering Automatic Control

Oriol Prats Vidal

Stockholm, 2010

Abstract

In a world where there are about 7 billion people living and this number is continuously increasing, it is necessary to reduce the resource consumption to achieve a sustainable situation.

In this work, a system to detect water and electricity consumption is introduced to identify resource waste and inform the consumers about it. For this purpose, many sensors are deployed in the kitchen of the department in order to gather data of the consumption. These sensors are an alternative to other ways of getting this information that are usually considered as invasive, like a camera. Even though, as far as there are many sensors deployed, the way to put all data together will be using wireless communication. Then, a treatment is performed in order to obtain some statistics about the consumption. These networks can also be accessed by people external to the department, who would be able to get information about their consumption and, consequently, their expenditures. Because all of this, a privacy problem of the enterprise appears, in this case the Electrical Engineering Automatic Control Department. Regarding the wireless communication, different protocols will be tested to check their reliability.

Furthermore, the data from the different sensors makes possible to determine behaviour patterns of the people working there. By using this system, not environmentally friendly actions that lead to resource waste can be easily identified and corrected. Again, this opens a discussion about privacy, concerning the different people and their habits.

To sum up, the system would enable the consumers to decrease their resource consumption, which would lead to an important energy saving in large scale and a reduction in costs in small scale.

Acknowledgments

First of all, I would like to thank my supervisors Karl H. Johansson and Jose Araujo for giving me all the support and attention they could and for answering to my doubts whenever I had them.

I would also like to thank Aitor Hernández, António Gonga and Aziz Khakulov for all his help during the development of this work and for being there every time I needed any of them.

To all my Erasmus colleagues, I am grateful to all of you for all the support you gave me during the whole year.

Also, thanks to everyone working on the Automatic Control department for letting me deploy all sensors and giving me advice for the work. It was really pleasant to see everyone interested in the experiment that was being developed.

Finally, thanks to my parents, who made possible to me this Erasmus experience and performing this work at KTH.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The continuous increase in resource consumption makes the government think about a way to turn it into a sustainable way for future generations. There is research in different topics to accomplish it, such as using renewable energy sources, minimizing the consumption or developing new techniques to improve the efficiency of the existing ones.

Research shows that fine grained resource monitoring combined with real-time feedback to the consumer motivates him to reduce the expenditure and try to reduce resource waste. In the EU residential and commercial buildings, the electrical consumption is about the 30% of the total, and the water consumed to generate it accounts for the 43% of the total water abstraction.[1][2] Even a small reduction in households will result in an important energy saving. Apart from energy, it will also produce financial benefits.

1.2 Problem formulation

As widely explained before, it is necessary to reduce the resource consumption on households. This thesis focuses on the resources consumption of the "Electrical Engineering Automatic Control" department of KTH University. In order to gather data of this consumption, a Wireless Sensor Network will be deployed (WSN) in it. A WSN consists of a network of wireless motes, which communicate with each other by radio, and each one of this motes are gathering data of the environment. Depending on which sensor is connected to the mote, it will gather different kinds of data. Vibration, current, light, temperature and infrared sensors will all be used in this thesis.

Concerning the communication, three different protocols will be implemented and tested to check their reliability. Specifically, they will use Carrier Sense Multiple Access (CSMA), Time Division Multiple Access (TDMA) and Carrier Sense Multiple Access with collision avoidance (CSMA-CA) medium access mechanism respectively.[3][4][5]

The purpose is to use all sensors to get the patterns in people behaviour since all the data will help to identify them and this will allow to provide people indication on how to do the things better, as well as statistics about the expenditures of the department.

1.3 Contributions

The main contributions of this thesis are:

- Implement an heterogeneous network of sensors in order to gather data of the consumption on different kinds of resources
- Treat this data to provide the consumer with feedback about his consumption
- Identify patterns in the behaviour to be able to inform the consumer about actions to reduce the resource consumption

1.4 Related work

Much work has already been performed in this area. Y. Kim et al[6] from Zhejiang University, gathered both electrical and water consumption from a household during 3 months. Their goal was to gather the data and identify the actions that were happening in the house. Another work, by X. Jiang et al[7] from University of California, Berkeley, consists on a WSN of 38 motes deployed all over an active laboratory and the analysis of the data from different points of view. Others, such as M. Chetty et al[8] from the Georgia Institute of Technology, Atlanta, focused their research on the environmental issue, specially the attitude that the consumer takes when he is aware of his consumption. They conclude that the consumer would take a position to do the right thing, but it helps if they are provided with real-time information of their resource consumption and its costs.

On another topic, Emmanuel Munguia Tapia[9], from MIT, focuses on the activity recognition and pattern detection with another objective. Patterns of behaviour are identified and transmitted

to doctors in order to correct non-healthy activities of daily living and correct them to improve the health of the patient. In this work, we will try to combine all this, gathering data from a flat, identifying patterns of behaviour and persuading the consumer to change his attitude about his expenditures.

Chapter 2

Smart Office: Wireless Sensor Network for Energy Monitoring and User Profiling

2.1 Smart grids and Home smart grids - energy consumption

The smart grid seems to be the solution to increase the efficiency of the energy usage. It is based on the communication between the producer, the distributor and the consumer of the energy in order to improve the quality of the service, avoiding blackouts and undesired events. The part of the smart grid that takes place in the consumer's house is called the home smart grid. The home smart grid consists on many sensors, actuators and routers deployed all around the house that collect the data and communicate with each other to assemble all the information.

The whole home smart grid communicates with the supplier of the energy and provides the consumer of information, such as the consumption and the energy price, this benefits the environmental and both the consumer and the supplier in financial concepts. Concerning the customer, today he receives a bill from the electricity enterprise once a month, or once every three months, with the whole energy consumption, and he does not get information about when or how he spent it. With the home smart grid, the customer will be aware of the time when he spends the energy and the cost it takes in every moment. With this information, he will be able to manage his energy consumption depending on many parameters, such as the price of the energy or the source it is

generating it. Also, the home smart grid will be able to automatically control some devices in order to optimize the energy consumption. For the supplier of the energy, it can get information about the trends of its clients, so it can connect or disconnect energy sources depending on the estimated demand. All this leads to a more efficient generation and consumption of the energy, reducing many issues concerning energy waste, e.g. the necessity of back-up plants when the demand is high or the usage of non-renewable sources.

2.2 Resource monitoring

To supply the department with a home smart grid, it is important to identify which devices and appliances are consuming more energy. Zimmermann analyzed the electricity consumption of 400 Swedish homes.[10] He shows the average annual consumption of different appliances. From his studies, it is possible to identify the appliances that consume the most. Table 2.1 shows the consumption of some appliances for a family between 26-64 years-old.

Appliance	**kWh/year**	**Appliance**	**kWh/year**	**Appliance**	**kWh/year**
Heating	8378	Clothes dryer	194	Kitchen stove	281
Water heating	2269	Dishwasher	193	Microwave	38
Fridge	275	TV	181	Water boiler	51
Fridge-freezers	489	Lighting	1021	Desktop	298
Clothes washer	209	Oven	175	Laptop	36

Table 2.1: Electrical consumption of some house appliances

Considering the devices available at the department and the data from table above, it was decided to place the sensors according to the structure in figure 2.1. It consists of a set of motes which collect data of the different sensors. To do it, the devices were monitored by using many different kinds of sensors.

The first ones, current sensors, were installed to monitor the electricity consumption of some devices, such as the microwave or the coffee machine. Then temperature sensors were also deployed. Their function was to monitor the temperature in the room, so the consumption on heating and air conditioning could be optimized.

Light sensors were also set up. With them, lights could be controlled to guarantee a good level

of light for working turning on and off the right lights. Also, infrared sensors were also installed in order to control the number of people in each one of the rooms. These ones, can work together with the temperature sensors, i.e. the heating could be lowered when many people enters the room, and with the light sensors, lights can be switched off when there is no one in the room.

Finally, vibration sensors were installed to monitor the ventilation. These ones were also used to monitor the water consumption. Also, the department is divided into different areas, the kitchen (at the bottom of the figure), the printing room (in the center) and the meeting and PhD students' room. This provides information about where the energy is used. For the communication, it was decided that these areas would have a separate nodes for each, so the number of motes communicating among each other would be decreased.

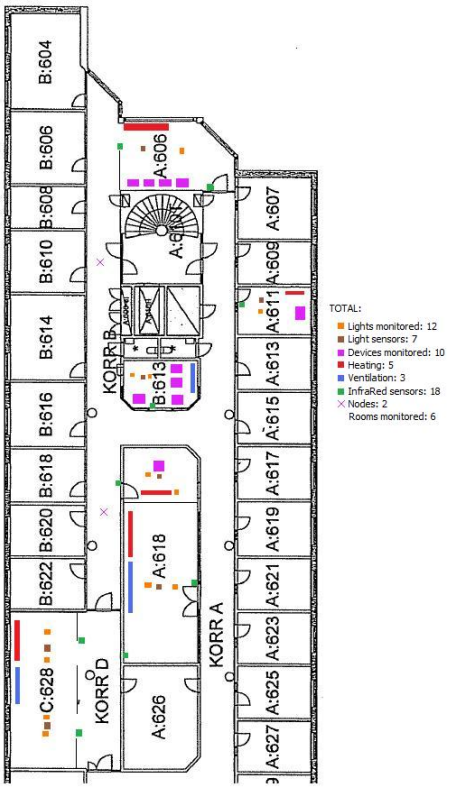
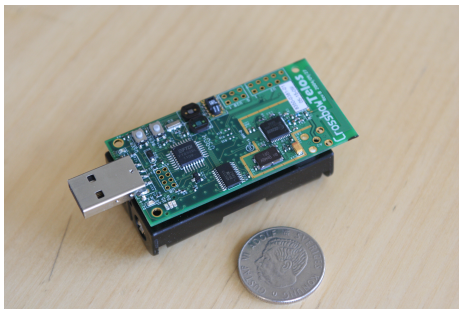


Figure 2.1: Planning of the sensors deployment

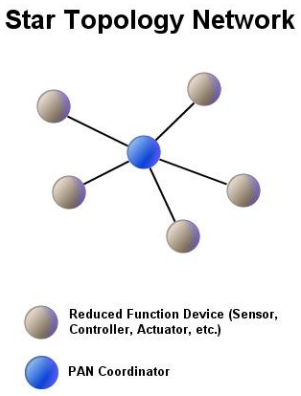
2.3 Wireless sensor networks

The WSN consists on a group of nodes which can be sensors, actuators, routers, etc. that communicate with each other to accomplish a function. In this case, the WSN is formed by different sensors, each one of them connected to a Telosb mote, which communicates with an intermediate mote, and this does the same with a base node. All sensors, intermediate motes and the base node consist of a Telosb mote connected to any kind of sensor. This typology is called a star network, where all motes in a group communicate just with their coordinator and, at the same time, all the motes coordinating each group communicate only with their own coordinator. Figure 2.2(a) shows an example of one of this motes compared with a 5 SEK coin and figure 2.2(b) shows this kind of network that is implemented. A Telosb mote is a technology that supports IEEE 802.15.4, contains many ADC channels and can be powered either by battery or USB. It already includes humidity, light and temperature sensors.[11]

To operate these motes TinyOS will be the operating system used. It is designed for WSNs, using an event-driven program of low-complexity. The language it uses is nesC, which is a dialect of C.[12]



(a)



(b)

Figure 2.2: (a) Picture of a Telosb mote, (b) Star network topology

2.4 Pattern recognition

Activities performed in the department can be identified from the data collected with the sensors. Then, from the whole activities it is possible to identify behaviour patterns and what is happening in the department. To do it, a Mealy machine will be implemented. It consists into a finite state

transducer that generates an output depending on the current state and the input. This implies that the state diagram includes both an input and an output for each transition edge.[13] The Mealy machine used for the pattern detection will be slightly modified from a typical one. As far as the activities are already identified, it will not use a clock, it will just move on with the activity following the last one. Figure 2.3 shows an example of the structure of this machine. The actions are the inputs that are entering to the machine and the states are the processes of what is happening in the kitchen. In the example, it could be interpreted as following:

- a is the action of someone entering the room, b someone leaving, c opening the fridge, d using the microwave and e taking a coffee
- *A, B, C* and *D* are the states of the machine and
- *ACTION 1, 2* or *3* are the outputs, when a pattern has been detected.

For example, following the diagram, if happens *a, c, e* and b , then it gets the output *ACTION 1/2*. From the events it means that someone entered, opened the fridge, took a coffee and left, and this can be interpreted as someone took a coffee with milk, this would be *ACTION 1*. The output can be *ACTION 1,2* depending on the events, if instead of happening *a, c, e* and b it got *a, e* and b it would be the coffee without milk, and it would be *ACTION 2*.

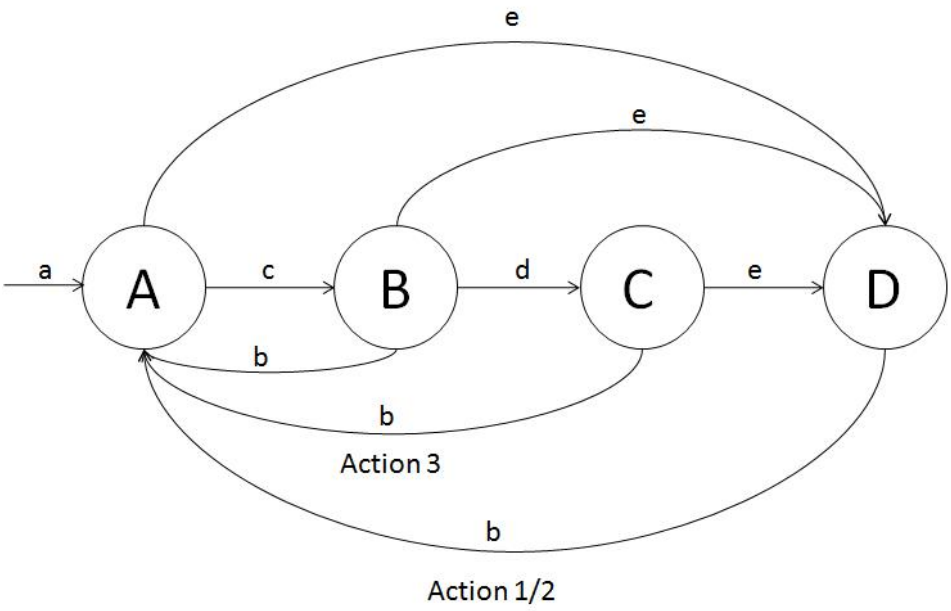


Figure 2.3: Example of Mealy machine algorithm used for pattern detection

Chapter 3

Smart Office Design and Implementation

3.1 Wireless Sensor Network

The deployment of the network was easy and quick. The order used was programming the motes with the correct program depending on the sensor, building the board to connect and power them from USB input, deploying a USB network in the kitchen to power all motes and place each one of them at their place.

3.1.1 Wireless Motes

Due to time reasons, the sensors deployment was only performed in the kitchen, so finally, the sensors that were installed were:

- 2 temperature sensors, one in each side of the room
- 2 light sensors, one for each light
- 2 vibration sensors, one for the water consumption and the other one for the fridge
- 4 current sensors, installed in two powerplugs, for the electrical devices
- 6 infrared sensors, divided in groups of 2 in each door, to control the number of people in the kitchen

Apart from these sensors, an intermediate mote and the base mote were also installed. Then, the whole installation was formed by 16 sensors and 11 motes. Figure 3.1 shows these deployment

and figure 3.2 shows the communication established among them.

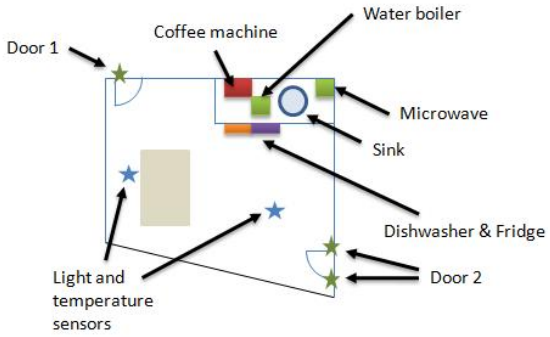


Figure 3.1: Deployment of sensors in the kitchen

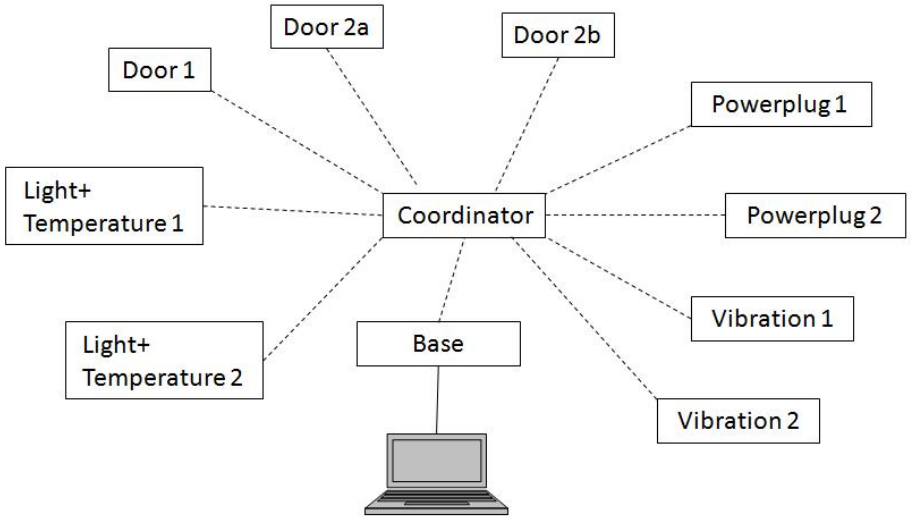


Figure 3.2: Communication in the wireless sensor network

3.1.2 Sensors

The sensors used to gather the data were the following ones. For light and the temperature, the sensors already integrated in the mote were Hamamatsu S1087[14] and Sensirion SHT11[15] respectively. Regarding the vibration, the Phidgets Vibration sensor[16] was deployed. For current,

also the sensors from Phidgets were used[17]. Finally, the infrared(IR) sensors used were the Sharp GP2Y0A21YK[18].

To receive their data, the Telosb motes use mainly the analog to digital converter(ADC) it has integrated. Then, just a transformation of the value is necessary to get the final data. Regarding the sensors that are already integrated in the mote, it already has some events implemented to gather their information.

Figure 3.3 shows the part of the code related to the data collection using the ADC. The first task, called "getData" asks for access to the ADC channels if it does not have it yet or launches a periodic timer if it already has it. The second event, called "Resource.granted", starts when the access to the channels is granted, if function "getData" asked for it, and it sets the right ADC channels that have to be monitored and launches the periodic timer. This timer is the third function in the code, called "Timer0", which calls the function of reading the data from the ADC channels, and when these data is ready, the last event starts, its function is to store the ADC values in variables and call its analysis. The analysis of the stored data is different depending on the sensor. All codes for each one of the sensors appear in the appendix. Hardware and the way that each sensor collects data are explained in the following sections.

```
void task getData()
{
      if (!call Resource.isOwner()){
              call Resource.request();
      }
      else {
              call Timer0.startPeriodic(50);

      }
}
event void Resource.granted()
{
  adc12memctl_t memctl[] ={{INPUT_CHANNEL_A0, REFERENCE_VREFplus_AVss},{INPUT_CHANNEL_A1,REFERENCE_VREFplus_AVss}};
  if (call MultiChannel.configure(&config, memctl, 2, buffer, 18, 500) == SUCCESS){
        call Timer0.startPeriodic(50);
  }
}

event void Timer0.fired()
{
  call MultiChannel.getData();
}

//when data is ready
async event void MultiChannel.dataReady(uint16_t *buf, uint16_t numSamples)
{
   atomic{
      data = buf;
      //IR1=(buf[5]+buf[8]+buf[11]+buf[14])/4;
      //IR2=(buf[4]+buf[7]+buf[10]+buf[13])/4;
      post AnalyseData();
   }
}
```

Figure 3.3: Code for the ADC channels reading

IR sensors

Concerning the IR sensors at the doors, they were installed in groups of 2 to be able to detect the direction of the person crossing the door. They were installed 2 on one door and 4, 2 on each side, on the other door. The reason for this was that the first door was about 1 meter wide, but the second one was about 1,5 meter wide, so the IR didn't enough range to detect everyone crossing. This was solved by installing 2 sensors on each side, so the range was wide enough. In case both devices detected the same person crossing, they were communicating with each other so one of them discarded that person.

Figure 3.4 shows these sensors together with the board that was built to power both the mote and the sensors and connect them. And figure 3.5 the schematics of the whole device. These sensors are gathering data distance with a sampling time of 50ms and analyzing this data with an algorithm in order to identify anything crossing the door. Then, they send a message every 10s with the number of people who has crossed that door. Finally, the base sums the numbers from the three IR sensors and gets the number of people inside the room.

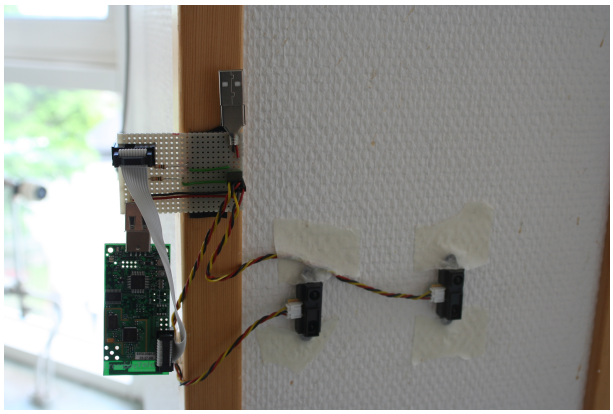


Figure 3.4: Picture of the IR sensors placed in one of the doors

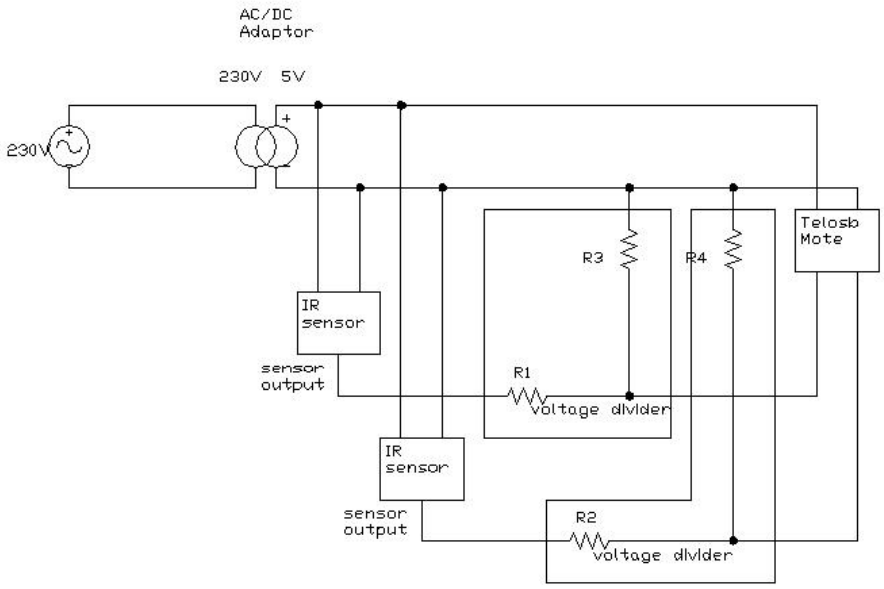


Figure 3.5: Schematics of the circuit for one of the IR sensors

Vibration sensors

About the vibration sensors for both the water consumption and the fridge, it was not possible to install them easily. The back of the fridge was not accessible to get data from the compressor, then finally it was installed on the door of the fridge, so it was possible to detect when someone opened or closed the fridge. With this detection, it is easier to identify the actions in the kitchen, and it is also a good way to detect the consumption, taking into account that the compressor works when the fridge is opened frequently and the cold escapes from inside. This is shown in figure 3.6 together with the board to connect everything. The schematics are shown in figure 3.7. These sensors are sampling the ADC channel every 100ms, then they save the highest value of every of second, and send a message every ten seconds with ten values, each one being the highest of each second.

Figure 3.6: Picture of the vibration sensor placed on the door of the fridge

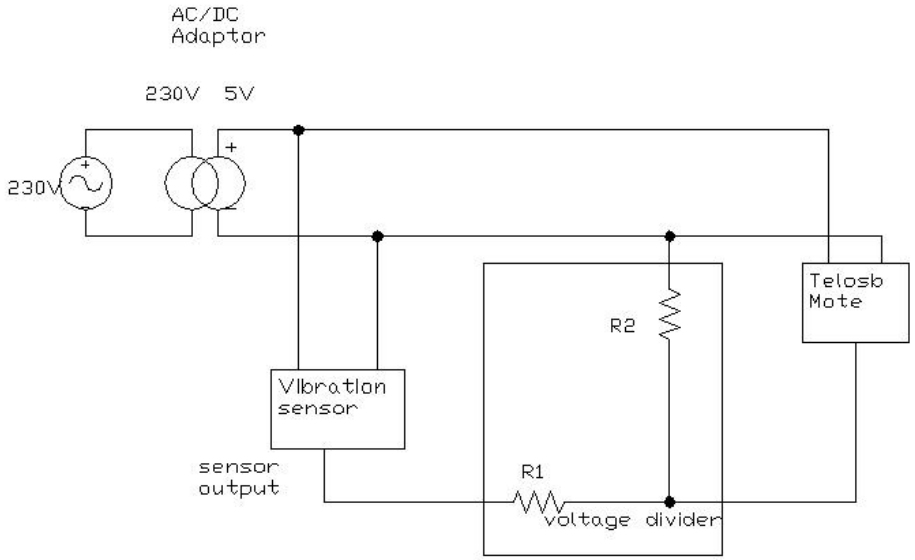


Figure 3.7: Schematics of the circuit for one of the vibration sensors

Concerning the other vibration sensor, it was not detecting the vibration of the water pipes, so in the end it was installed at the bottom of the sink. Then it was able to detect the vibration of the sink when the water hit it, and also when someone placed a cup or any other object on the sink. Figure 3.8 shows this deployment as well as the connecting board. The schematics is the same as the other vibration sensor and it also works the same way.

Light and temperature sensors

Telosb motes already include both light and temperature sensors. For this reason, data from these sensors are both gathered by the same mote. To get reliable information of the light, these motes are placed facing the light that are being monitored. Their sampling times are 10s for the light sensors and 1min for the temperature sensor. The reason for this is that the temperature changes

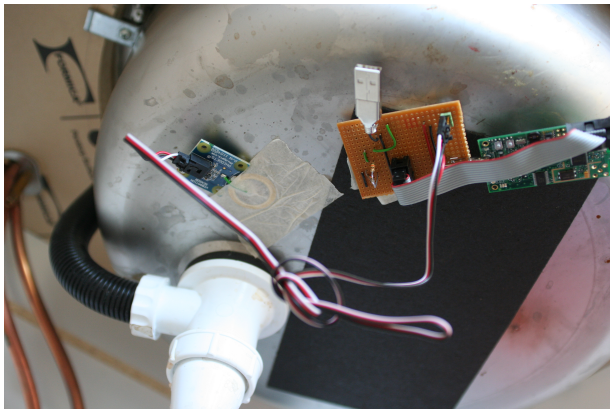


Figure 3.8: Picture of the vibration sensor placed below the sink

much slower than the rest of parameters monitored. These motes are the only ones that are not connected to the USB through the board. It is because they already have the sensors integrated so they do not need to be powered independently and it is possible to connect the mote directly to the USB. Figure 3.9 shows one of these motes.

Figure 3.9: Picture of the light and temperature sensor placed facing the lights

Powerplug

The powerplug is the device that integrates two current sensors to gather data from both plugs it has installed, so it is possible to connect two devices in each powerplug. This measuring device was created by Kiu Liu[19]. For this thesis, the same device was used, except that the program installed in the mote was a different one.

The powerplugs were installed to measure the current from the water boiler, the coffee machine, the dishwasher and the microwave. Also, two more powerplugs were installed eventually to make some tests. The powerplug gathering data from the coffee machine and the water boiler samples data every 100ms for the first plug and every 1s for the second one. The data sent in the packets are the sum of all values in the last 10 seconds for the coffee machine and the ten different values for the water boiler. The other one, gathering data from the dishwasher and the microwave, it works for both the same way it does for the water boiler, it samples every second and sends all these values in the packet.

It is possible to see this powerplug and the schematics of the circuit in figures 3.10 and 3.11.

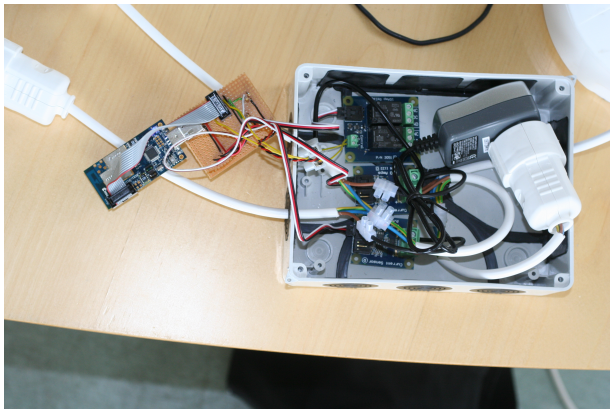


Figure 3.10: Picture of one of the powerplugs

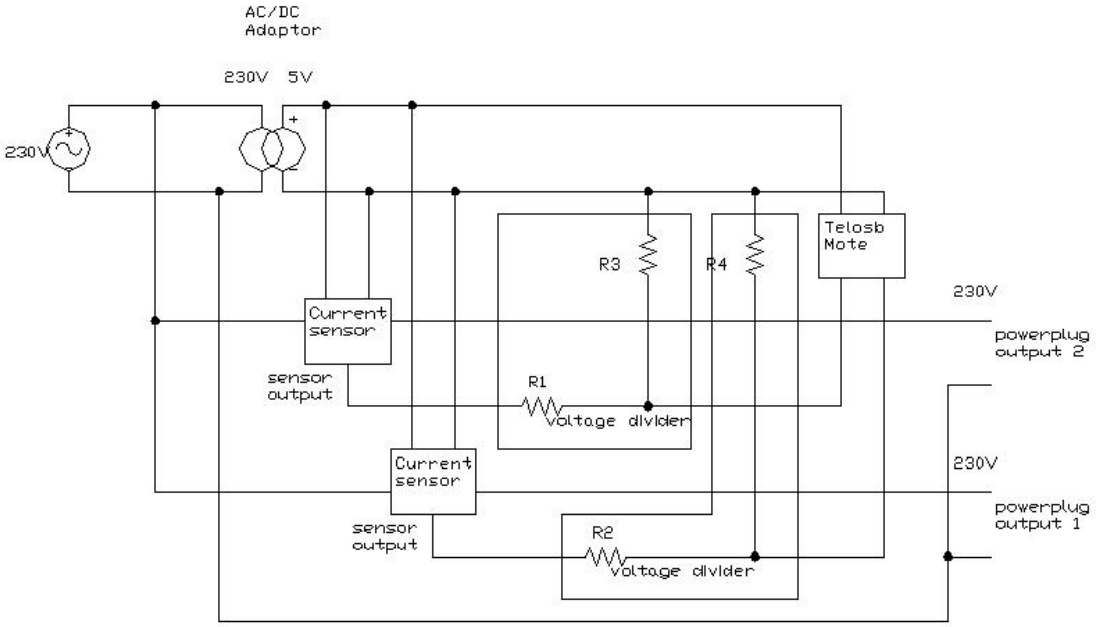


Figure 3.11: Schematics of the circuit for one of powerplugs

3.2 Communication protocol

Telosb technology uses a CC2420[20] radio transceiver to communicate. As explained in section 1.2, the communication was implemented using three different methods. The first one was using CSMA, the second one with TDMA and the last one with CSMA-CA.

All of them used the same packet structure, which is shown in figure 3.12, as well as some constants, like the radio channel used for the communication or the identification number of the base and the coordinator mote.

```
#ifndef __APP_PROFILE_H
#define __APP_PROFILE_H

enum {
  RADIO_CHANNEL = 25,
  BASE_NODEID=10,
  INTERMEDIATE_NODEID=11,
  AM_MYMSG = 10, //Radio channel number
  BUFFER_SIZE=10,
  FIRST_PLUG=1,
  SECOND_PLUG=2,
};

//structure of message
typedef nx_struct MyMsg{// max 28 bytes
   nx_uint8_t other;
   nx_uint8_t srcId;
   nx_uint16_t trgtId;
   nx_uint16_t data[BUFFER_SIZE];
}MyMsg;

#endif
```

Figure 3.12: Structure of the packet

CSMA is already implemented in TinyOS, so it was just programming the motes with the correct events. The code for this protocol is the same as for TDMA, just removing the part related to the synchronizing packets and the time slots. An example of the CSMA protocol is shown in 3.14, just do not consider the lines that correspond to TDMA.

Regarding TDMA, a slot was assigned to each of the sensors when they could transmit, and a syncronyzing signal was sent from the coordinator every 10 minutes. By this method, the messages were received staggeredly one after the other by the base mote. During the rest of the time that the motes are not allowed to transmit, they can switch to sleep mode in order to save energy. The time slots structure is shown in figure 3.13.

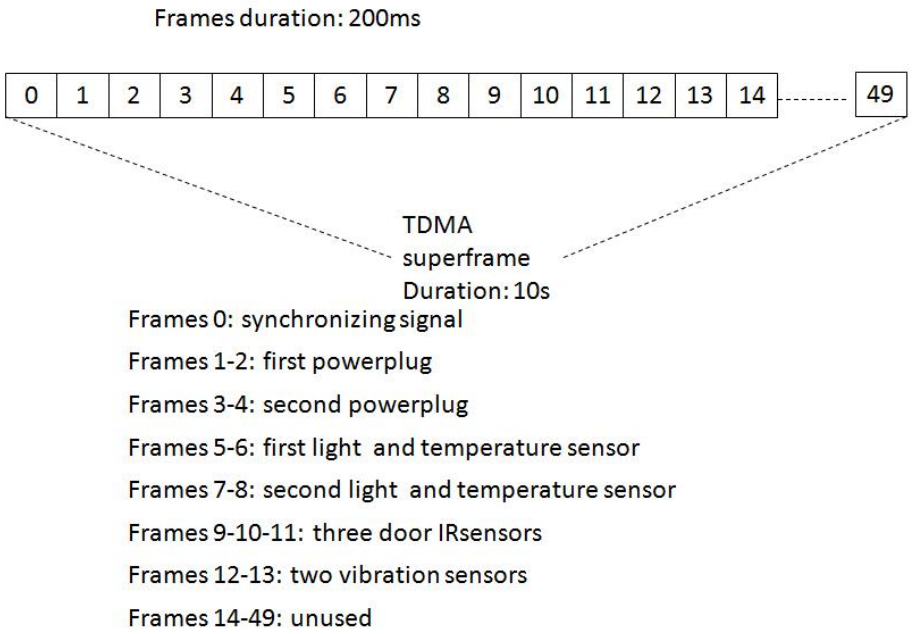


Figure 3.13: Time slots structure for the TDMA communication

Figure 3.14 shows an example of the code for the communication for the TDMA protocol for one of the sensors. The second function, called "Receive.receive", is to analyze a packet when the mote receives one. It checks if the mote is the destination and acts in consequence. If the packet is the synchronizing signal, the mote restarts all timers and launches a new one depending on the time slot that it has assigned, then it just talks in its own slot until a new synchronizing packet is received. Also, the packet received can also be a signal to change the sampling period of the sensor.

```
  task void sendMessage(){
      if(!busy) {
        MyMsg* datapkt =(MyMsg*)(call Packet.getPayload(&pkt,sizeof(MyMsg)));
        datapkt->srcId = TOS_NODE_ID;

                for (j=0;j<BUFFER_SIZE;j++) {
                   datapkt->data[j] = maxvalues[j];
                }
                datapkt->trgtId = INTERMEDIATE_NODEID;
                datapkt->other = 0;

        if(call AMSend.send(AM_BROADCAST_ADDR, &pkt,sizeof(MyMsg))==SUCCESS){
                call Leds.led2Toggle();
                busy = TRUE;
                for (j=0;j<BUFFER_SIZE;j++){
                    vibration[j]=0;
                    maxvalues[j]=0;
                }
                k=0;
        }
      }
  }

  //when a message is received
  event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len) {
     if (len == sizeof(MyMsg)) {
        MyMsg *receivepkt = (MyMsg*)payload;
        if(receivepkt->srcId==BASE_NODEID){

//************ TDMA  protocol *******
//the synchronising signal is identified with target mote of 111 and the field other equal to 3

           if(receivepkt->other==3 && receivepkt->trgtId==111){
              call Timer1.stop();
              call Timer2.stop();
              i=0;
              call Timer0.startOneShot(2000);
           }

//***********************************

           else if(receivepkt->other==4 && receivepkt->trgtId==TOS_NODE_ID){
              Ts=receivepkt->data[0];
           }
        }
        return msg;
     }
  }

  //finish sending a message
  event void AMSend.sendDone(message_t* msg, error_t error) {
    if (&pkt==msg) {
    busy = FALSE;
      //Sets that the radio is not busy
    }
  }

  event void AMControl.startDone(error_t err) {
    if (err == SUCCESS) {
       call Timer1.startPeriodic(100);
       call Timer2.startPeriodic(Ts);
    }
    else {
    call AMControl.start(); //The radio has not been started. It tries again to start.
    }
  }

  //When the radio has been turned off successfully
  event void AMControl.stopDone(error_t err) {
  }
```

Figure 3.14: Time slots structure for the TDMA communication

Finally, for the CSMA-CA, it was designed by Aitor Hernández and tested in this work[21]. It is based on the IEEE 802.15.4 Standards and it is a beacon-enabled network. This communication protocol is a typical star-type network where the central node coordinates all the communication among all nodes.

The time frame of the protocol is called superframe, and all nodes of the network has to follow its structure when they want to send a packet. This superframe is bounded by two consecutive beacons sent by the coordinator node. It is also divided in two parts, the active portion and the inactive portion. In the first the node is allowed to send messages, and in the second one, the mote goes to sleep mode in order to save energy. The active portion is also divided in two parts, the Contention Access Period (CAP) and the Contention Free Period (CFP). The first one is for CSMA-CA communication protocol and the second one is divided again in time slots that can be assigned for any of the motes to send as many packets as wanted. This mechanism of assigning time slots is called Guarantee Time Slots(GTS). At the beginning of each superframe, all nodes should receive the corresponding beacon, which contains all information concerning the following superframe. Figure 3.15 shows an image with the structure of this superframe[22]. In the figure, k is the k superframe, *S.D.* is the superframe duration, which is divided in 16 equally sized time slots, *B.I* the beacon interval and $\bar{T}_k$ is the time when the superframe begins.

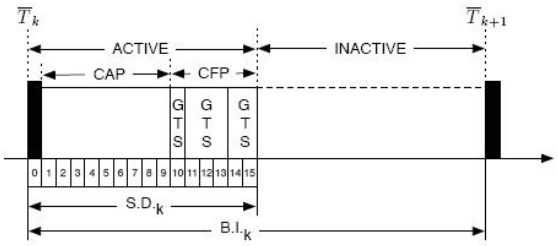


Figure 3.15: Superframe time organization

In this work, the packets are sent on the CAP of the superframe, no GTS is used, so the communication is based on CSMA-CA in the IEEE 802.15.4 Standard. Figure 3.16 shows an example of the code for this communication protocol for one of the sensors.

```
/*****************************************************
* IEEE 802.15.4
*****************************************************/

void startApp()
{
  ieee154_phyChannelsSupported_t channelMask;
  uint8_t scanDuration = BEACON_ORDER;

  call MLME_SET.phyTransmitPower(TX_POWER);
  call MLME_SET.macShortAddress(TOS_NODE_ID);

  // scan only the channel where we expect the coordinator
  channelMask = ((uint32_t) 1) << RADIO_CHANNEL;

  // we want all received beacons to be signalled
  // through the MLME_BEACON_NOTIFY interface, i.e.
  // we set the macAutoRequest attribute to FALSE
  call MLME_SET.macAutoRequest(FALSE);
  m_wasScanSuccessful = FALSE;
  call MLME_SCAN.request  (
                           PASSIVE_SCAN,           // ScanType
                           channelMask,            // ScanChannels
                           scanDuration,           // ScanDuration
                           0x00,                   // ChannelPage
                           0,                      // EnergyDetectListNumEntries
                           NULL,                   // EnergyDetectList
                           0,                      // PANDescriptorListNumEntries
                           NULL,                   // PANDescriptorList
                           0                       // security
                         );
}

event void MLME_RESET.confirm(ieee154_status_t status)
{
  if (status == IEEE154_SUCCESS)
    startApp();
  }

  event message_t* MLME_BEACON_NOTIFY.indication (message_t* frame)
  {
    // received a beacon frame
    ieee154_phyCurrentPage_t page = call MLME_GET.phyCurrentPage();
    ieee154_macBSN_t beaconSequenceNumber = call BeaconFrame.getBSN(frame);

    if (!m_wasScanSuccessful) {
      // received a beacon during channel scanning
      if (call BeaconFrame.parsePANDescriptor(
            frame, RADIO_CHANNEL, page, &m_PANDescriptor) == SUCCESS) {
        // let's see if the beacon is from our coordinator...
        if (m_PANDescriptor.CoordAddrMode == ADDR_MODE_SHORT_ADDRESS &&
            m_PANDescriptor.CoordPANId == PAN_ID &&
            m_PANDescriptor.CoordAddress.shortAddress == COORDINATOR_ADDRESS){
          // yes! wait until SCAN is finished, then syncronize to the beacons
          m_wasScanSuccessful = TRUE;
        }
      }
    } else {
      // received a beacon during synchronization, toggle LED2
      if (beaconSequenceNumber & 1)
        call Leds.led2On();
      else
        call Leds.led2Off();
    }

    return frame;
  }
```

```
event void MLME_SCAN.confirm    (
                          ieee154_status_t status,
                          uint8_t ScanType,
                          uint8_t ChannelPage,
                          uint32_t UnscannedChannels,
                          uint8_t EnergyDetectListNumEntries,
                          int8_t* EnergyDetectList,
                          uint8_t PANDescriptorListNumEntries,
                          ieee154_PANDescriptor_t* PANDescriptorList
                        )
{
  if (m_wasScanSuccessful) {
    // we received a beacon from the coordinator before
    call MLME_SET.macCoordShortAddress(m_PANDescriptor.CoordAddress.shortAddress);
    call MLME_SET.macPANId(m_PANDescriptor.CoordPANId);
    call MLME_SYNC.request(m_PANDescriptor.LogicalChannel, m_PANDescriptor.ChannelPage, TRUE);
    call MLME_SET.macRxOnWhenIdle(TRUE);
    call Frame.setAddressingFields(
          &m_frame,
          ADDR_MODE_SHORT_ADDRESS,        // SrcAddrMode,
          ADDR_MODE_SHORT_ADDRESS,        // DstAddrMode,
          m_PANDescriptor.CoordPANId,     // DstPANId,
          &m_PANDescriptor.CoordAddress,  // DstAddr,
          NULL                            // security
          );

          post getData();
          call Timer1.startPeriodicAt(2400,Ts);
    } else
    startApp();
}

event void MCPS_DATA.confirm    (
                          message_t *msg,
                          uint8_t msduHandle,
                          ieee154_status_t status,
                          uint32_t timestamp
                        )
{
  if (status == IEEE154_SUCCESS && m_ledCount++ >= 2) {
    m_ledCount = 0;
    call Leds.led1Toggle();
  }
 // previous
  //post packetSendTask();
}

event void MLME_SYNC_LOSS.indication(
                          ieee154_status_t lossReason,
                          uint16_t PANId,
                          uint8_t LogicalChannel,
                          uint8_t ChannelPage,
                          ieee154_security_t *security)
{
  m_wasScanSuccessful = FALSE;
  call Leds.led1Off();
  call Leds.led2Off();
  call Timer0.stop();
  call Timer1.stop();

  startApp();
}

event message_t* MCPS_DATA.indication (message_t* frame)
 {
   if (call Frame.getPayloadLength(frame) == m_payloadLen && call Frame.getFrameType(frame) == FRAMETYPE_DATA) {
       MyMsg *receivepkt = (MyMsg*) (call Packet.getPayload(frame,m_payloadLen ));

   if(receivepkt->srcId==BASE_NODEID){

       if(receivepkt->other==3 && receivepkt->trgtId==111){
               call Leds.led0Toggle();
               call Timer1.stop();
           }
           else if(receivepkt->other==4 && receivepkt->trgtId==TOS_NODE_ID){
              Ts=receivepkt->data[0];
           }
     }
   }
   return frame;
 }
```

Figure 3.16: Part of the code related to the communication for CSMA-CA protocol

For the communication from the computer to the motes, another program in matlab was developed by Aitor Hernández[21]. This program creates a packet and forwards it through the serial port to the base node and it sends it to the network.

3.3 WSN Deployment

As stated in section 3.1.1, the network follows a star topology. In this deployment, the network is formed by all the motes together with their sensors communication to their coordinator mote, and this last one communicating with the base node, connected to the computer. This structure can be seen in previous figure 3.2.

The messages are sent by the sensors following their communication protocol and their sampling time, then the coordinator mote receives and forwards them to the base, that forwards them through the serial port of the computer. Finally, these messages are gathered using Simulink

and analyzed by matlab in order to get all the information. The Simulink model receives the messages that the base mote is sending over the serial port and stores them in variables in matlab. Figure 3.17 shows the simulink model used to gather the messages.

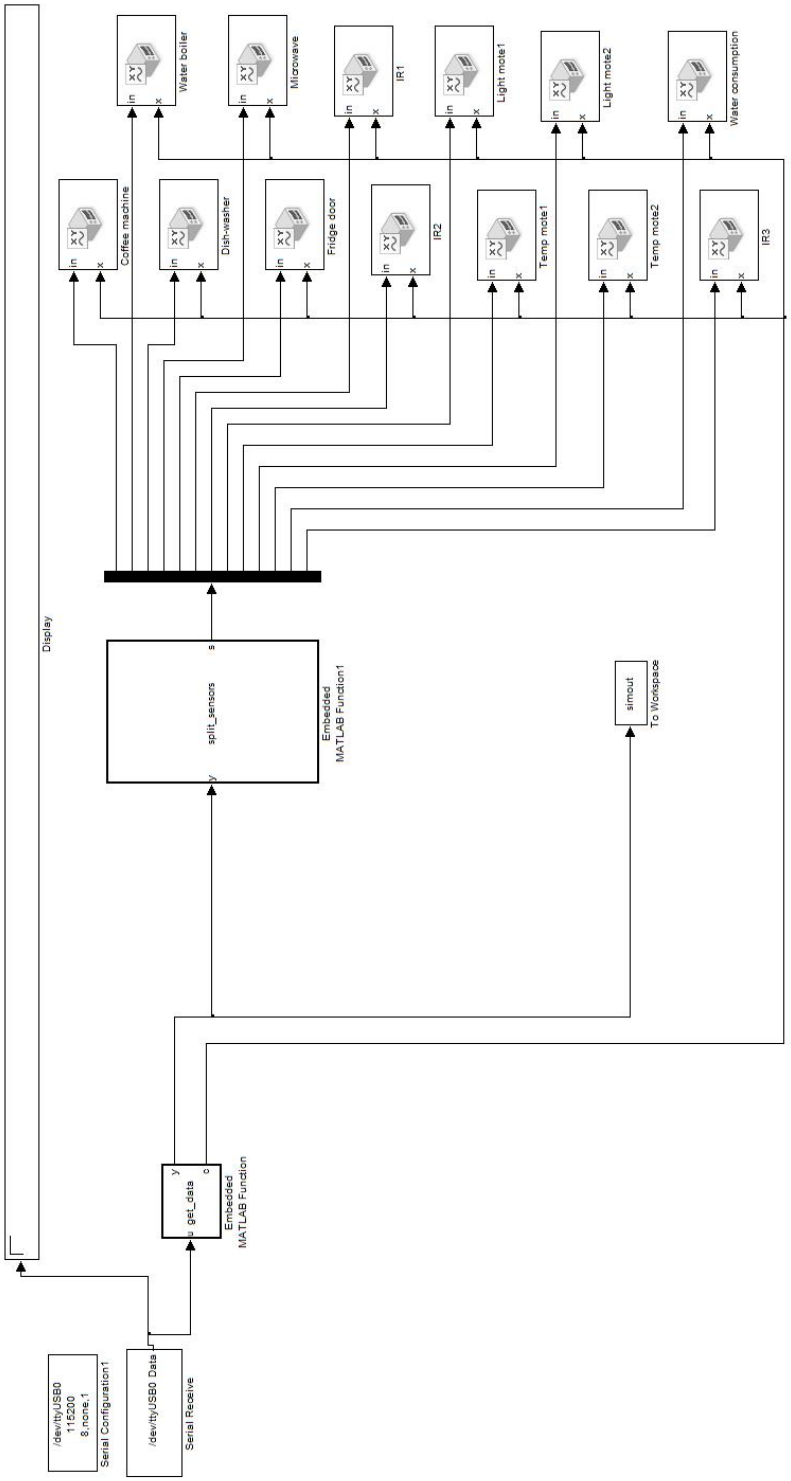


Figure 3.17: Simulink model used to receive the messages over the serial port

3.4 Events and pattern recognition

Next step would be to identify the events that are happening from the data gathered. To do it, an analysis is performed by matlab. For most of the sensors, it just consists into a threshold value, when the data gets over this threshold, an event is identified. For others, it is based in a change on the value. All events identified are registered into a matrix for each sensor, that contains time and a binary number, which is 1 when it was higher than the threshold and 0 when it was not. For sensors that can have three different states, such as the fridge that can be opened or closed or the sink that can be water or something different, the value -1 is also used in the matrix. Regarding the IR sensors, the matrix contains the number of people counted inside the kitchen. The next matrix shows an example of a part of a matrix for one of the sensors.

$$\text{microwave} = \begin{pmatrix} ... & 1213,107 & 1213,134 & 1213,162 & 1213,190 & 1213,218 & 1213,245 & ... \\ ... & 0 & 0 & 0 & 1 & 1 & 1 & ... \end{pmatrix}$$

Figure 3.18 shows the code for the creation of one of these matrices. "a" is a matrix with all the data gathered corresponding to one of the sensors, and "b" is another one containing only the time and the values received. Then, "sink" is the binary matrix, in this case it is tertiary because it can also contain a -1. The code for all the matrices is included in the appendix.

```
%for the sink vibrational sensor
%we will plot a 1 when someone is using water and a -1 when someone
%puts a cup in the sink

k=0;
j=1;
l=0;

if (a(1,2)==0) && (a(1,3)==28)
    clear sink
    for i=1:lengthB
        if b(i,1)>3600 || b(i,1)<2800  %these numbers are thresholds
            sink(j,2)=-1;
            sink(j,1)=b(i,2);
            j=j+1;
        elseif 3400>b(i,1) && b(i,1)>3300
            sink(j,2)=0;
            sink(j,1)=b(i,2);
            j=j+1;
        elseif (3600>b(i,1) && b(i,1)>3400) || (2800<b(i,1) && b(i,1)<3300)
            sink(j,2)=1;
            sink(j,1)=b(i,2);
            j=j+1;
        end
    end
    [lengthSink,widthSink]=size(sink);
end
```

Figure 3.18: Matlab code for creating matrices containing the data

The threshold values for the events are:

- For the coffee machine, an event is registered when the value exceeds 200.
- Regarding the rest of the current sensors for the devices, the events are recognized when the value gets over 4A.
- For the vibration sensor below the sink, the usual output value is around 3350, and this value is disturbed depending on what is done on the sink. Important peaks, about higher than 3600 or lower than 2800, are interpreted as something left on the sink, and the rest of oscillations, between 2800 and 3300 and between 3400 and 3600, which seems just noise, is water falling on the sink, which creates just a small vibration, depending of the quantity of water falling on it.
- About the other vibration sensor, placed on the door of the fridge, spykes above 3400 are supposed to be the events when the door is opened or closed, as well as when it goes below 3300.

- For light and temperature sensors, no thresholds were implemented because there was no use of them in the months of monitoring.
- Finally, concerning the IR sensors, the events are identified when there is a change on the value of people inside the kitchen.

Then, the matrices from all sensors are put together into a bigger. As far as there are different sampling times for each sensor, the process is done depending on the time that was also stored. For the slower sampling times, there will be many empty slots between two consecutive values, which will be filled in with the first one. As a result, a matrix containing the binary values for each one of the sensors will be obtained, all of them ordered by time. Time is a sexagesimal system, so it is converted into a linear system to plot it properly, which will still be stored. Next matrix shows an example of a part of the matrix for all sensors.

$$\mathrm{M} = \begin{pmatrix} ... & 1213,359 & 1213,368 & 1213,396 & 1213,423 & 1213,451 & 1213,479 & ... \\ ... & 0 & 0 & 0 & 0 & 0 & 0 & ... \\ ... & 0 & 0 & 0 & 1 & 1 & 1 & ... \\ ... & 1 & 1 & 0 & 0 & 0 & 0 & ... \\ ... & 0 & 0 & 0 & 0 & 0 & 0 & ... \\ ... & 0 & 0 & 0 & 0 & 0 & 0 & ... \\ ... & 0 & 0 & 1 & 1 & -1 & 0 & ... \\ ... & 2 & 2 & 3 & 3 & 3 & 3 & ... \end{pmatrix}$$

Figure 3.19 shows a part of the code to create this big matrix. The whole code is included in the appendix.

```
%%% CREATE THE MATRIX WITH ALL THE DATA IN IT %%%%
% we will save the time in the first row and then the data from the
% diferent sensors in each following row. To do this we'll be checking the
% time of each data and saving it in the matrix in a position that the time
% fits with an error of less than 1 second. We have sensors that their data
% is every second and other that are every 10 seconds, so we will fill all
% the time positions in the middle with the last real value of the sensor.

if(lengthPLOT==9)
    clear M

    M=boiler';
    i=1;

    for j=1:lengthCoffee
        if coffee(j,1)<2399.5
            while abs(coffee(j,1)-M(1,i))>2399 && i<lengthBoiler
                i=i+1;
            end
            while (coffee(j,1)-M(1,i))>0.015 && i<lengthBoiler
                if(j>1)
                    M(3,i)=coffee(j-1,2);
                end
                i=i+1;
            end
            if (abs(coffee(j,1)-M(1,i)))<=0.015 && i<lengthBoiler
                M(3,i)=coffee(j,2);
                i=i+1;
            end
        end
    end

    i=1;

    for j=1 lengthMicrowave
       if microwave(j,1)<2399.5
            while abs(microwave(j,1)-M(1,i))>2399 && i<lengthBoiler
                i=i+1;
            end
        while (microwave(j,1)-M(1,i))>0.015 && i<lengthBoiler
            if(j>1)
                M(4,i)=microwave(j-1,2);
            end
            i=i+1;
        end
        if (abs(microwave(j,1)-M(1,i)))<=0.015 && i<lengthBoiler
            M(4,i)=microwave(j,2);
            i=i+1;
        end
       end
    end
    ⋮
    ⋮
    ⋮
```

Figure 3.19: Matlab code for creating the matrix with all the events registered

The matrix "microwave" represents the binary values of the device related to the time. For the matrix "M", it includes the time and the values for all sensors, being the rows for water boiler, coffee machine, microwave, dishwasher, fridge, sink and people respectively from row 2 to 8.

Apart from the matrix, matlab also outputs a log with the actions that have occurred and the time for each one. To do this, and also calculate some statistics about the consumption and the people behaviour, some other matrices are created too. Figures 3.20 and 3.21 show the code for it. In the figures, and also in diagram in figure 3.22, the letters represent the events that are happening, they make the current state, the numbers, to change. The letters correspond to each event as the following:

- E: someone enters the room
- L: someone leaves the room
- c: coffee machine switched on
- e: coffee machine switched off
- b: water boiler switched on
- v: water boiler switched off
- m: microwave switched on
- i: microwave switched off
- d: dishwasher switched on
- h: dishwasher switched off
- f: fridge opened
- g: fridge closed
- w: water turned on
- s: someone put something on the sink

```
%%% CREATE THE EVENTS VECTOR FROM THE MATRIX %%%

% we will create three vectors that contain the sequence of all the events
% that happened. then just checking the order of these events we can guess
% what was going on in the kitchen

j=1;
clear E %vector of events
clear T %vector with the time of each corresponding event
clear P %vector with the number of people in the room at each event

for i=2:widthM

    if M(2,i)>M(2,i-1)
        E(j)='b';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    elseif M(2,i)<M(2,i-1)
        E(j)='v';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    end

    if M(3,i)>M(3,i-1)
        E(j)='c';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    elseif M(3,i)<M(3,i-1)
        E(j)='e';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    end

    if M(4,i)>M(4,i-1)
        E(j)='m';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    elseif M(4,i)<M(4,i-1)
        E(j)='i';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    end

    if M(5,i)>M(5,i-1)
        E(j)='d';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    elseif M(5,i)<M(5,i-1)
        E(j)='h';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    end

    if M(6,i)>M(6,i-1)
        if M(6,i)~=0
            E(j)='g';
            T(j)=M(1,i);
            P(j)=M(8,i);
            j=j+1;
        end
    elseif M(6,i)<M(6,i-1)
        if M(6,i)~=0
            E(j)='f';
            T(j)=M(1,i);
            P(j)=M(8,i);
            j=j+1;
        end
    end

    if M(7,i)>M(7,i-1)
        if M(7,i)~=0
            E(j)='w';
            T(j)=M(1,i);
            P(j)=M(8,i);
            j=j+1;
        end
    elseif M(7,i)<M(7,i-1)
        if M(7,i)~=0
            E(j)='s';
            T(j)=M(1,i);
            P(j)=M(8,i);
            j=j+1;
        end
    end

    if M(8,i)>M(8,i-1)
        E(j)='E';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    elseif M(8,i)<M(8,i-1)
        E(j)='L';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    end
end

[lengthE,widthE]=size(E);
```

Figure 3.20: Code for showing an output log with the events and get some statistics

```
%%% WE SHOW ALL THE EVENTS THAT HAPPENED AND COUNT SOME STATISTICS %%%

nb=0;
nc=0;
nm=0;
nd=0;
nw=0;
ns=0;
ne=0;
nl=0;
nf=0;

for j=1:widthE
    if E(j)=='b'
        disp(sprintf('At %d:%d boiler started',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        nb=nb+1;
    elseif E(j)=='v'
        disp(sprintf('At %d:%d boiler finished',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
    elseif E(j)=='c'
        disp(sprintf('At %d:%d coffee machine started',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        nc=nc+1;
    elseif E(j)=='e'
        disp(sprintf('At %d:%d coffee machine finished',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
    elseif E(j)=='m'
        disp(sprintf('At %d:%d microwave started',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        nm=nm+1;
    elseif E(j)=='i'
        disp(sprintf('At %d:%d microwave finished',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
    elseif E(j)=='d'
        disp(sprintf('At %d:%d dishwasher started',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        nd=nd+1;
    elseif E(j)=='h'
        disp(sprintf('At %d:%d dishwasher finished',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        nd=nd+1;
    elseif E(j)=='w'
        disp(sprintf('At %d:%d water',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        nw=nw+1;
    elseif E(j)=='s'
        disp(sprintf('At %d:%d sink',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        ns=ns+1;
    elseif E(j)=='f'
        disp(sprintf('At %d:%d fridge opened',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        nf=nf+1;
    elseif E(j)=='g'
        disp(sprintf('At %d:%d fridge closed',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
    elseif E(j)=='E'
        disp(sprintf('At %d:%d entered',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        ne=ne+1;
    elseif E(j)=='L'
        disp(sprintf('At %d:%d left',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        nl=nl+1;
    end
end
```

Figure 3.21: Code for showing an output log with the events and get some statistics

Finally, using the method explained it section 2.4, patterns on the people behaviour will be identified from the data gathered. To do it, the data will be analysed by another program in matlab. The way this program works is explained in the diagram in figure 3.22 and a part of the code is shown in figure 3.23.

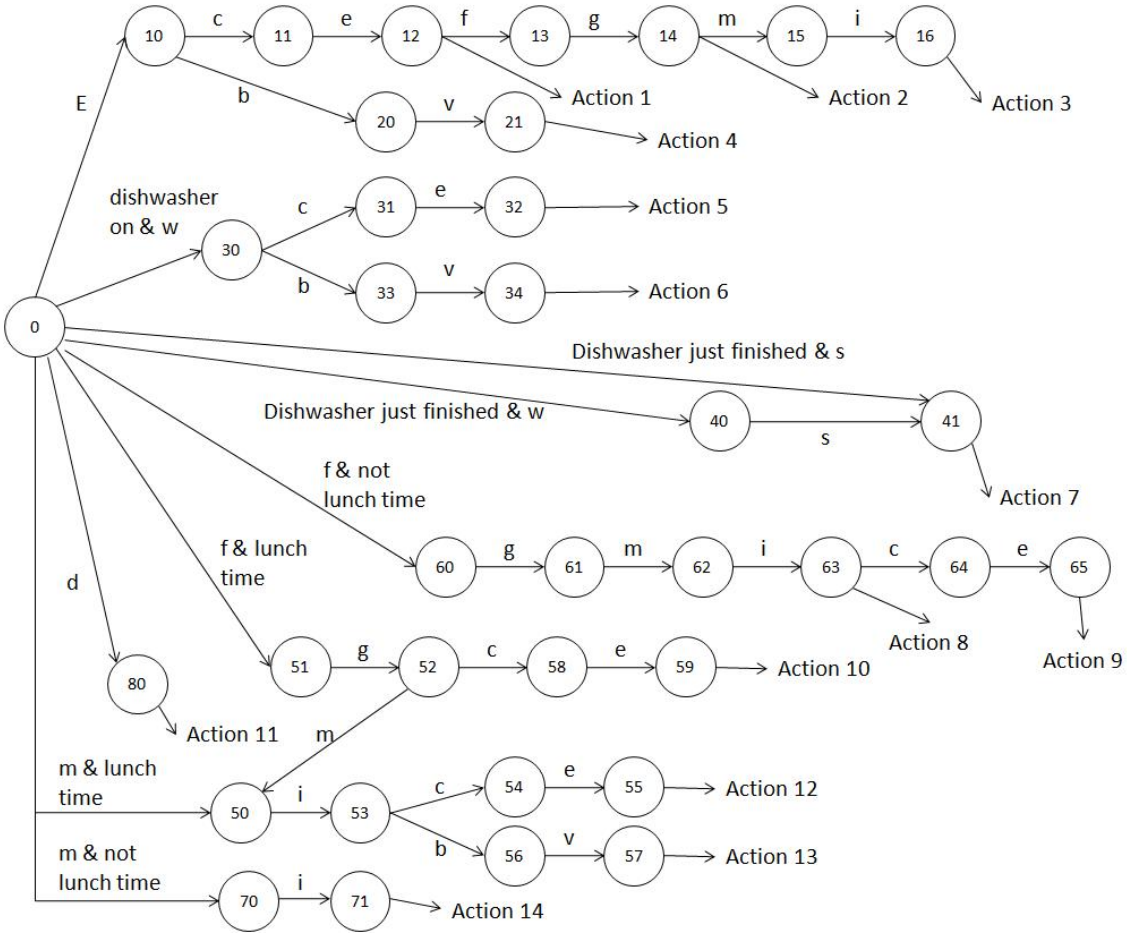


Figure 3.22: Algorithm for pattern recognition

```
%%% IDENTIFY THE PATTERNS FROM THE EVENTS VECTOR %%%

current_state=0;
DW_ON=0;
dishw=0;
warmwater=0;
fika=0;
DWstart=0;

for j=1:widthE

    if T(j)>1150 && T(j)<1400
        lunch_time=1;
    else
        lunch_time=0;
    end

    if E(j)=='v'
        warmwater=T(j);
    elseif E(j)=='d'
        DW_ON=1;
        if T(j)>(DWstart+50)
            DWstart=T(j);
        end
    elseif E(j)=='h'
        dishw=T(j);
        DW_ON=0;
    end

    if current_state==0
        if E(j)=='E'
            current_state=10;
        end

    elseif current_state==10
        if E(j)=='c'
            current_state=11;
        elseif E(j)=='b'
            current_state=20;
        elseif E(j)=='d' && (T(j)-DWstart)==0
            current_state=80;
        elseif E(j)=='L' && (T(j)>warmwater && T(j)<(warmwater+50))
            disp(sprintf('At %d:%d someone entered, may have taken tea with water that was already warm and left',
              fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
            current_state=0;
        elseif DW_ON==1 && E(j)=='w'
            current_state=30;
        elseif DW_ON==0 && E(j)=='w'&& (T(j)>dishw && T(j)<dishw+50)
            current_state=40;
        elseif DW_ON==0 && E(j)=='s'&& (T(j)>dishw && T(j)<dishw+50)
            current_state=41;
        elseif lunch_time==1 && E(j)=='m'
            current_state=50;
        elseif lunch_time==1 && (E(j)=='f' || E(j)=='g')
            current_state=51;
        elseif lunch_time==0 && (E(j)=='f' || E(j)=='g')
            current_state=60;
        elseif lunch_time==0 && E(j)=='m'
            current_state=70;
        end
        ⋮
```

Figure 3.23: Part of the code for analyze the events and detect the behaviour patterns

There are plenty of actions that can be identified by using this method. The only ones that can happen apart from the shown in the diagram are the detection of a "fika" break or cake break. These actions are identified when there is at least a determined number of people in the room and it is around 3 in the afternoon.

All this analysis is performed considering actions done by only one person, for this reason, the current state will always go back to 0 each time that someone leaves the room.

Chapter 4

Evaluation

4.1 Data Analysis

The deployment and installation of the network was done during the month of May, so all the data shown was gathered during June and July. In Sweden, during these months it's very clear so it's not necessary to switch the lights on during the day, and it's also warm enough not to switch the heaters on. For this reason, there's no significant data regarding electric consumption for light and heating.

4.1.1 Statistics regarding electrical consumption

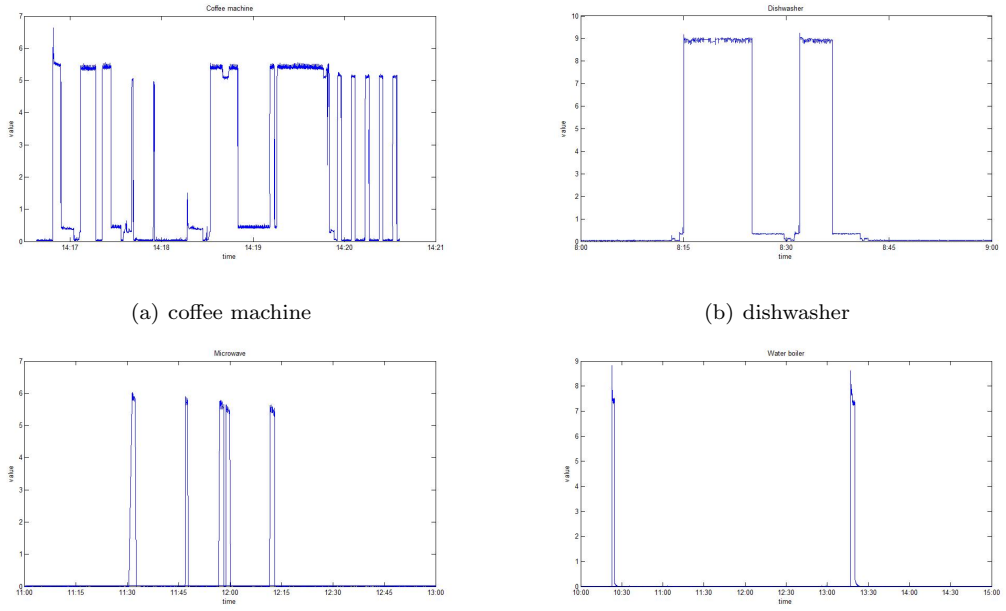
From all the data gathered, it is possible to get some statistics about the energy consumption of the department. Table 4.1 shows the values of the electric consumption of the devices.

Appliance	**Usage per day**	**Consumption per Use**	**per day**
Coffee machine	19	0.01 kWh average for all kinds of coffees	0.19 kWh
Water boiler	5	0.086 kWh water tank at 75% of capacity	0.43 kWh
Microwave	8	0.025 kWh half power and 1min working	0.2 kWh
Dishwasher	1	0.52 kWh for economy program at 60º C	0.52 kWh

Table 4.1: Electric consumption of the monitored devices

All these values were calculated from different groups of data. Figure 4.1 shows some plots of the devices consumption. The values for the consumption per use were calculated from these groups of

data. Figure 4.1(a) is for the coffee machine, and it shows the consumption of these device when an espresso and a normal coffee were taken. The sampling time in this sample is 50ms. Figure 4.1(b) shows the consumption of the dishwasher for an economy program wash. This program consumes electricity during two block of time, with a while in the middle that does not consume, but it is still the same washing program. Figure 4.1(c) shows the electricity consumed by the microwave during a 2h interval. It is easy to identify the 5 times that people warmed their food in it, as far as it is lunch time. Finally figure 4.1(d) shows the consumption of the water boiler. It just shows that someone had tea at about 10:30 and someone else after lunch. For all the last three plots the sampling time is 1s.



(a) coffee machine

(b) dishwasher

(c) microwave

(d) water boiler

Figure 4.1: Electrical consumption of (a) coffee machine, (b) dishwasher, (c) microwave and (d) water boiler

Table 4.2 shows some other interesting statistics related to people behaviour.

Statistic	**Mean**	**Max**	**Min**
Number of coffees per day	19	26	7
Number of times the water boiler is used	5	9	2
Number of times the microwave is used	8	13	3
Number of times the dishwasher is used	1	2	0
Number of people who has lunch there	6	8	2
Number of times people puts milk in the coffee	11	12	8
Occupancy of the kitchen	3	23	0
Statistic	**Earliest**	**Latest**	**Max peak**
Coffee machine	about 8:00h	about 17:00h	at 13:00*
Water boiler	about 10:00h	about 16:00h	at 13:00h
Microwave	about 8:00h	about 13:00h	at 12:00h
People entering	about 5:30h	about 20:00h	at 15:00
People arriving for lunch	about 11:30h	about 14:00h	at 12:00

**or at 15:00h when cake*

Table 4.2: Statistics of people behaviour

Another interesting plot is in figure 4.2, which shows the data gathered from the coffee machine during a whole working day. It is possible to see that the machine is switched off and it has a low consumption until the first person arrives and takes a coffee. Then it keeps a higher consumption while people is taking coffees, and it switches off automatically in the afternoon after a while that no coffee has been taken.

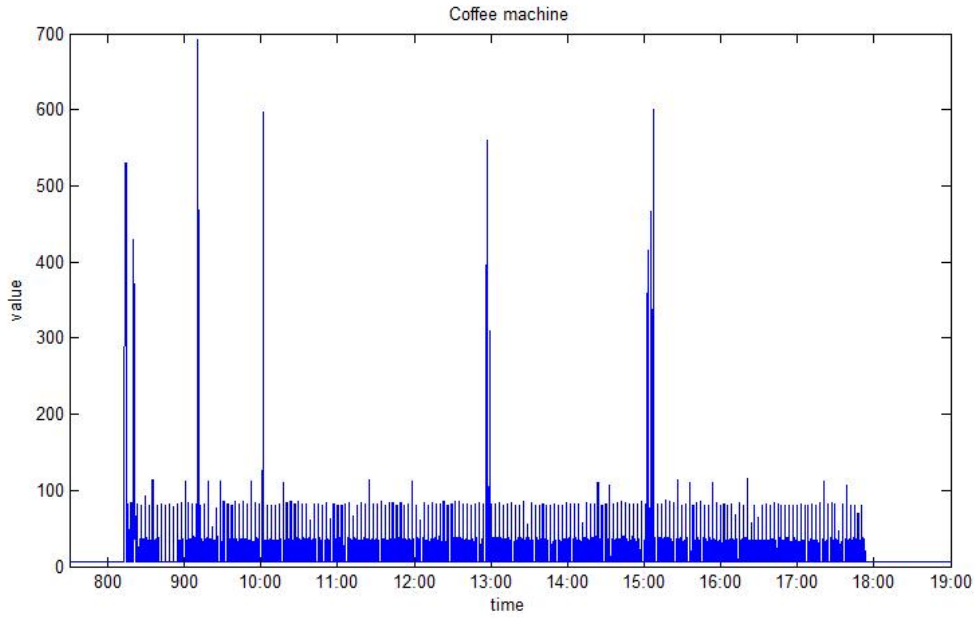


Figure 4.2: Coffee machine consumption during a whole day

The rest of devices do not have this kind of function. The microwave just waits on standby until someone uses it, the dishwasher is switched on or off by the user by it does not do it automatically, and the water boiler can only be on or off, it does not have standby position.

4.1.2 Vibration sensor below the sink

Apart from the electrical consumption of the devices, other resources were also measured. Water was one of this ones. Figure 4.3 shows the water consumption during a whole day. The vertical axe represent the raw output value from the mote, a voltage in a scale from 0 to 4095. The way to interpret the plot is explained in section 3.4. This method just makes possible to identify what is happening, but it is not possible to get a value about the quantity of water consumed. Anyway, it can show non-environmentally friendly patterns, such as washing cups by hand instead of placing them in the washing machine. It also makes possible to identify someone not following the rules of the department. It is stated that when the dishwasher is finished, the first person who arrives has to empty it and put the dirty cups in it again. If someone washes his cup just after the dishwasher is finished, there is someone who is not following the rules.

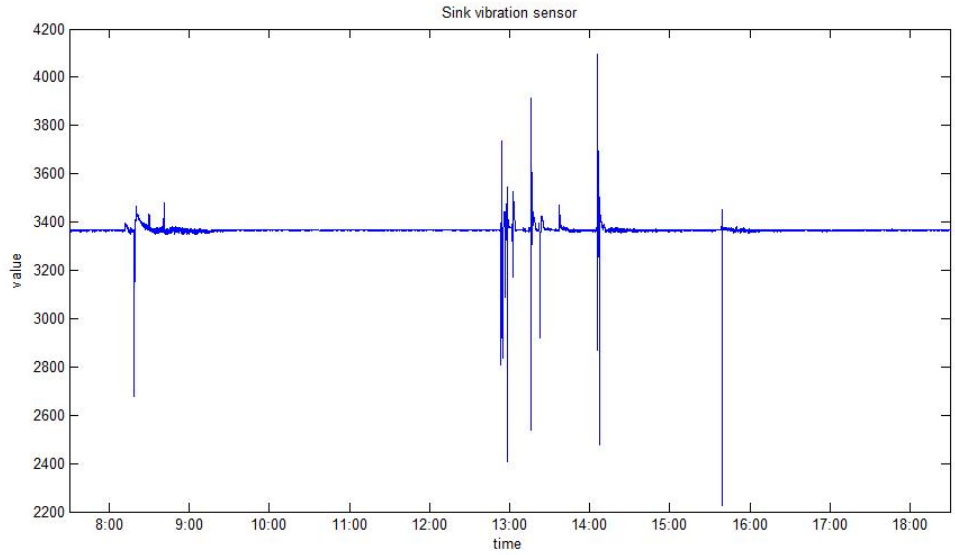
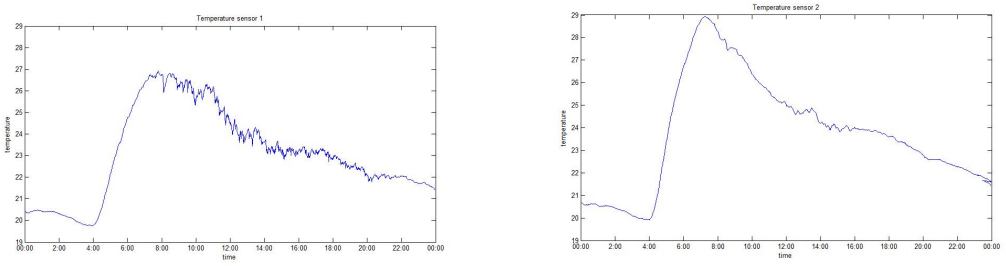


Figure 4.3: Water consumption during a whole day

4.1.3 Temperature sensors

As explained in the beginning of the section, heating was not used because it was not necessary. Anyway, data from the temperature was measured. Figures 4.4(a) and 4.4(b) show the temperature gathered by the two temperature sensors.



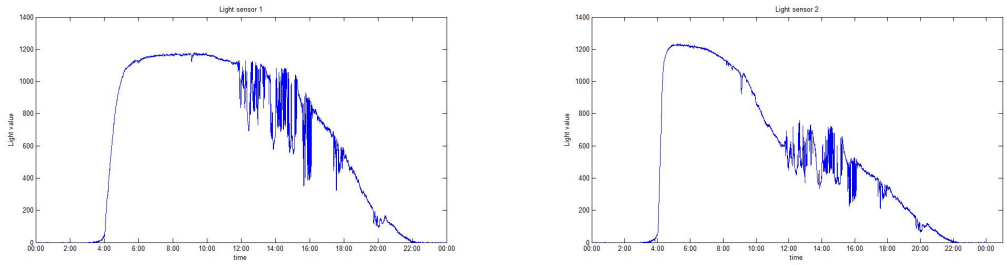
(a) Temperature sensor 1

(b) Temperature sensor 2

Figure 4.4: Temperature registered by (a) temperature sensor 1 and (b) temperature sensor 2

4.1.4 Light sensors

Light was also measured by the sensors. Again, there is no interesting data about it because it was not necessary to use them. The light sensors were placed facing the lights, so in case the lights are switched on the value increases suddenly, so it is easy to identify when they are on or off. Figures 4.5(a) and 4.5(b) show the data from both light sensors. The sunrise was at about 4am and it reaches soon the highest value. Then it just keeps decreasing until it is totally dark again. An intermediate output value from the sensor that would mean light enough to work without the necessity of switching the lights on would be around 700, always depending on the person and his preferences.



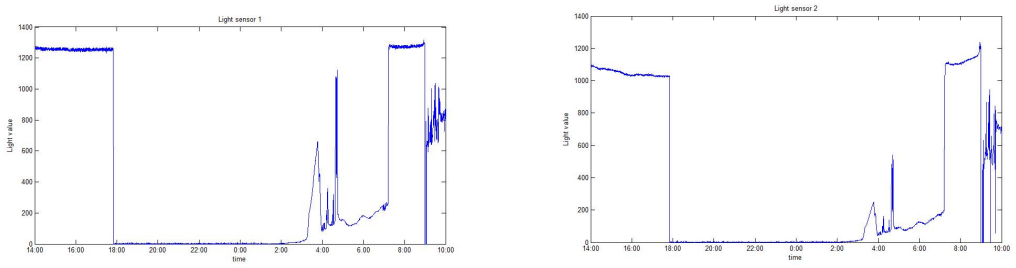
(a) Light sensor 1 (b) Light sensor 2

Figure 4.5: Light value registered by (a) light sensor 1 and (b) light sensor 2

Just to show how to identify the light consumption, some plots were taken in the beginning of the experiment, when it was being deployed. The data for figures 4.6(a) and 4.6(b) is from the beginning of May. It is possible to see how the lights are turned off at about 18h, when it is totally dark. Then the sunrise starts at 4am but it is much slower than before. Then the lights are switched on again at about 7am, when the first person enters in the kitchen, and switched back off at about 9am. It is possible that at 9am the lights are not necessary so they are turned off to save energy.

4.1.5 Vibration sensor on the fridge door

Regarding the vibration sensor on the fridge, it was placed on the door as explained in section 3.1.2 and it was not possible to get the electrical consumption neither. Anyway it makes possible to identify when the door is opened or closed in order to detect people behaviour. Figure 4.7 shows the data get from these sensor during a whole day. The way to analyze it is explained in section 3.4. There is a lot of noise so it is hard to identify it properly. Apart from this, usually the vibration is higher when the door is closed than when it is opened, so the small spykes are



(a) Light sensor 1

(b) Light sensor 2

Figure 4.6: Light value registered by (a) light sensor 1 and (b) light sensor 2 when lights were switched on

supposed to be the door opened and the big ones the door closed. It is possible to see that the fridge is opened mainly at 12h, the lunch time, and 13h, to take out milk for the coffee. There are other spykes representing people taking or placing anything in the fridge or taking the milk too.

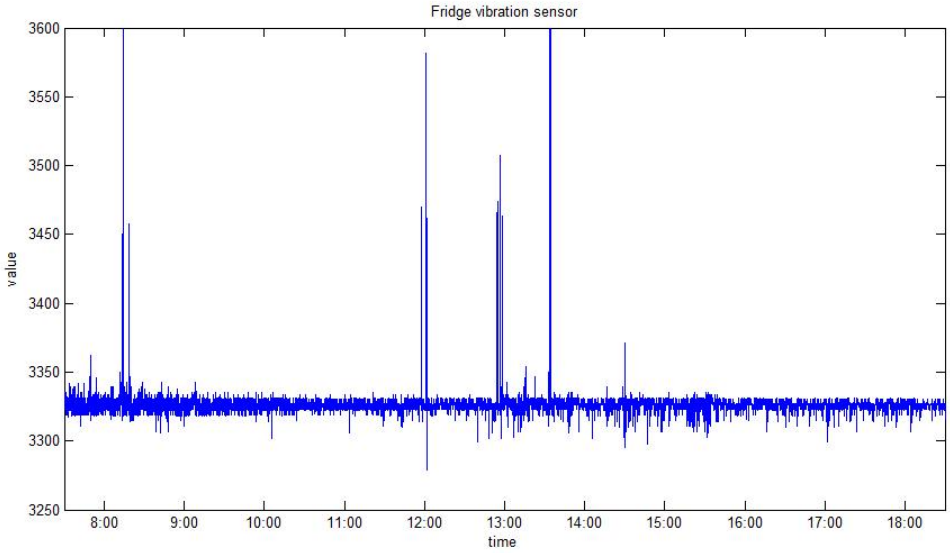


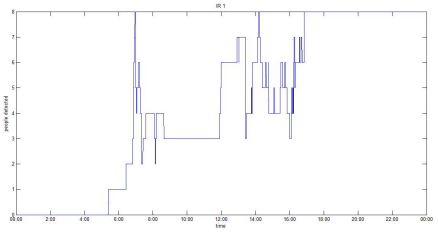
Figure 4.7: Fridge activity during a whole day

4.1.6 IR sensors

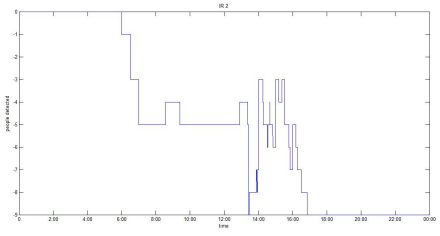
Finally, IR sensors were collecting data about people entering or leaving the kitchen in order to identify their actions and try to detect patterns in their behaviours.

Also, these sensors provide information about the number of people who is inside the kitchen. By treating these data, together with the temperature by the other sensors, it would make possible to achieve conclusions about the usage of heating or air conditioning, i.e. when the kitchen is warm enough and many people enters, the heating could be lowered to reduce the consumption, as far as there are many people in the room the temperature will remain comfortable. The motes gather data about people entering or leaving and store the value and send it every 10s, so it is possible that there is more than 1 person of difference in two consecutive packets. Also, it is possible that the number of people entering and leaving of a mote during a day it is not the same, as far as people are not forced to leave by the same door that they entered, the numbers that must be the same are the sum of people entering and leaving of all three motes. Even though, this system of counting people is not perfect and it does not work the 100% of times, so it is easy to be a small difference between the two numbers.

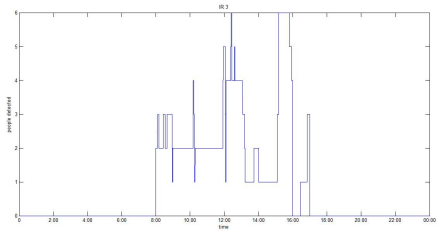
Figure 4.8 shows data gathered by these IR sensors about the number of people in the kitchen during a whole day. IR 1 corresponds to the mote placed on the narrow door, and IRs 2 and 3 are the other ones placed on the wide door. It is possible to observe that the first person arriving does it at about 5:30h, and the last one leaving is at about 17:00h. It must be pointed out that this is the time concerning the kitchen, not the whole department. It can also be observed that many people is entering and leaving the kitchen at about 8:00 or 9:00, when people go to have a coffee or tea, or between hours like 12:00h and 15:00h, when people go for lunch or "fika".



(a) IR 1



(b) IR 2



(c) IR 2

Figure 4.8: People detected by (a) IR 1, (b) IR 2 and (c) IR 3

4.1.7 Interconnection of powerplugs

Another analysis of the devices was also performed. The main idea was to try to monitor more devices with less powerplugs. To do it, devices and plugs were connected in other ones so a lot of data was gathered together and repeated times. Figure 4.9 shows the connection of the devices. For example, powerplug 1 monitors the coffee machine and the water boiler separately, and powerplug 2 monitors them together. If it were possible to identify the consumption of each one in powerplug 2, we could monitor two devices with just one plug, and we would have the other one empty to monitor another device.

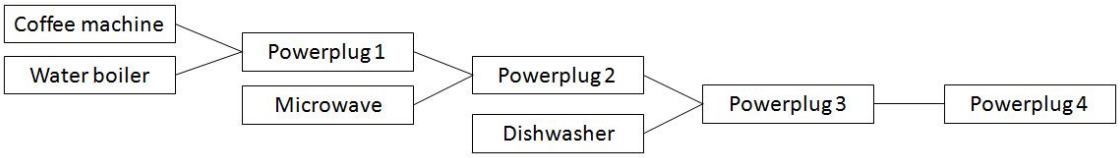


Figure 4.9: Powerplugs connections to gather mixed data

Figure 4.10 shows the plots of the data gathered from all these plots. It is possible to see that the values from sensors measuring more than one device correspond to the values measured by the plug measuring only that device. There are many errors too, it is mainly due to packet loss, with four plugs there were many more messages exchanged and the packet loss was much higher. Anyway, it is quite easy to identify the different devices in the plots were they are together. The coffee machine are just spykes and about 5 amperes, while the water boiler is during a little longer and about 8 amperes. Regarding the microwave, it is quite similar to the coffee machine, it uses about the same current and for or less the same time. Finally, for the dishwasher, it consumes current for two long periods of time, so it is the easiest to identify. In conclusion, it would be possible to connect altogether the water boiler, the dishwasher and either the coffee machine or the microwave, but not both, because are the ones easy to confuse. To do it, it is also necessary that the current sensor supports the sum of current of the devices, in this case the current sensor is up to 30A, and the sum of the maximum consumption of the three devices is about 18A, so it would be feasible.

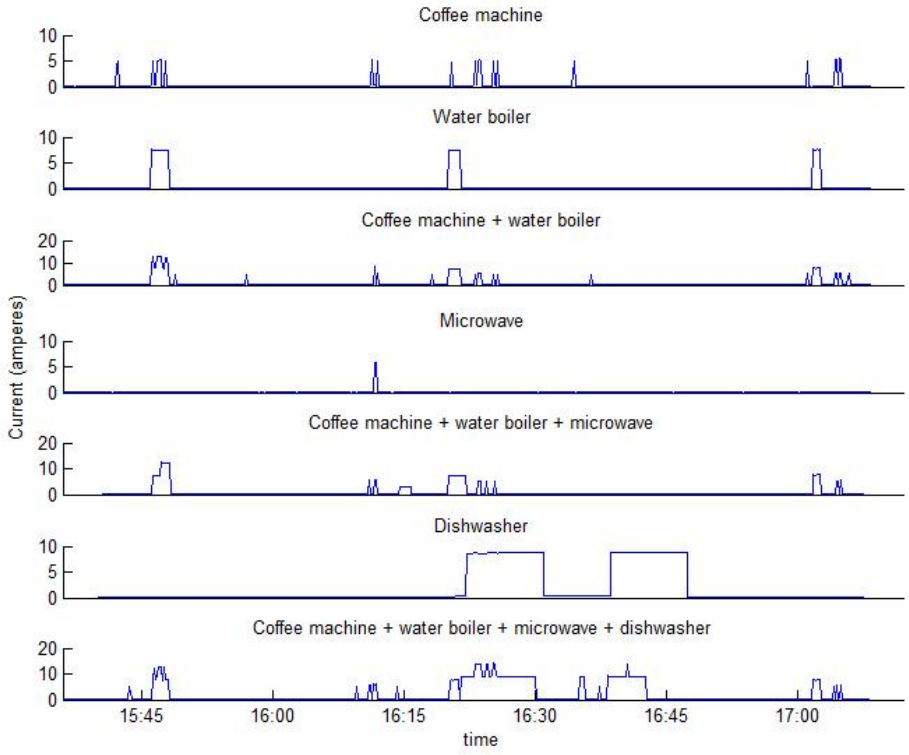


Figure 4.10: Electrical consumption of the devices when the powerplugs were interconnected

4.2 Communication analysis

Concerning the communication among all motes, three different protocols were used to control it, as explained in section 3.2. Some analysis were performed to determine the reliability of each one of these protocols. In this application, only the packet losses are interesting. The delay does not affect the performance of the system. In order to carry out these experiments, the data was treated in matlab with many programs. First, data from all protocols was analysed with a simple program that calculates the packet loss from the difference of time between two consecutive packets. If this difference is larger than the sampling time, a new packet is created in between with the values of the last packet received. In case that many consecutive packets are lost, all of them will be created with these last values received. Table 4.3 shows the packet loss statistics for the three protocols.

Using CSMA protocol, there are important losses in many of the motes, probably due to coincidence when they send the packet to the intermediate mote. IR 2 has a really big loss of packets, which

Sensor	**Packets send**	**Packets lost**	**% of packets lost**
Powerplug 1	17736	1065	6,00
Powerplug 2	17740	743	4,19
Vibration fridge	8866	362	4,08
Vibration sink	8847	1399	15,81
Light sensor 1	8866	377	4,25
Light sensor 2	8866	483	5,44
Temperature sensor 1	1481	72	4,86
Temperature sensor 2	1479	86	5,81
IR 1	8866	383	4,32
IR 2	6132	2179	35,53
IR 3	8867	418	4,71
TOTAL	97746	7567	7,74

Table 4.3: Packet losses with CSMA protocol

cannot be explained by just communication problems, it must have been due to a hardware problem. Anyway, the rest of motes still have an important loss of packets. The most common problem is that two motes send their packets to the intermediate mote. This one receives both packets, but it only forwards the last one to the base mote, so even that we received both messages, only one is forwarded. In order to try to solve this, the TDMA protocol was tested, obtaining better results shown in table 4.4.

Sensor	**Packets send**	**Packets lost**	**% of packets lost**
Powerplug 1	17673	1586	8,97
Powerplug 2	17749	97	0,55
Vibration fridge	8872	50	0,56
Vibration sink	8925	339	3,80
Light sensor 1	8877	51	0,57
Light sensor 2	8872	123	1,39
Temperature sensor 1	1476	17	1,15
Temperature sensor 2	1475	19	1,29
IR 1	9154	46	0,50
IR 2	8879	114	1,28
IR 3	8875	47	0,53
TOTAL	100827	2489	2,47

Table 4.4: Packet losses with TDMA protocol

It is possible to see that powerplug 1 has the largest losses. This can be due to the position of the mote, because it is placed in the powerplug box, that is in the cupboard under the sink. With this new protocol the packet loss has decreased significantly from a 7,74% to a 2,47%. Even though, this is still a big loss that should be reduced. A loss of a 2,47% with only 9 motes sending packets could be much more important if some more motes were added to the network, which would be the main point to build a whole home smart grid. To try to do it, the last protocol was implemented. As stated in section , it was designed by Aitor Hernández[21], so the new communication code was integrated to the existing one. The analysis of this protocol was done in another way. A wireless sniffer was used to do it. It consists in a device that logs all wireless traffic over the network and decodes and analyses its content. The sniffer used was CC2420 Development Kit[23], from Texas Instrument, and the software was formed by two parts. The first one, which its function was to get the data that the sniffer was collecting, was SmartRF Packet Sniffer[24], also from Texas Instrument. And the second one, which treated these data and calculated the statistics was based in matlab, and it was also designed by Aitor Hernández[21]. With this matlab program, much more statistics about the communication are calculated. Also, the analysis is done depending on the mote identification number, so about the motes sending both temperature and light values, the results will be of both data combined, and it will also get information about the intermediate mote. Table 4.5 shows the results.

Sensor	**Packets send**	**Packets lost**	**% of packets lost**
Intermediate mote	2256	1	0,04
Powerplug 1	413	0	0
Powerplug 2	407	0	0
Vibration fridge	204	0	0
Vibration sink	202	0	0
Temperature and light sensor 1	237	0	0
Temperature and light sensor 2	237	0	0
IR 1	203	0	0
IR 2	207	0	0
IR 3	202	0	0
TOTAL	4568	1	0,02

Sensor	**Retransmitted**	**% of retransmission**	**time between packets**
Intermediate mote	5	0,22	876.5ms
Powerplug 1	23	5,57	4598.7ms
Powerplug 2	1	0,25	4859.1ms
Vibration fridge	1	0,49	9717.4ms
Vibration sink	0	0	9765.3ms
Temperature and light sensor 1	1	0,42	8317.2ms
Temperature and light sensor 2	0	0	8358.6ms
IR 1	0	0	9765.6ms
IR 2	12	5,80	9488.7ms
IR 3	0	0	9765.5ms
TOTAL	43	0,93	7551.3ms*

**mean time between packets for one mote*

Table 4.5: Packet losses with CSMA-CA protocol

At first glance it is observed that the packet loss has decreased considerably to an only 0,02%. Apart from this, the probability to retransmit a packet is about 1%, lower than the packet loss experienced before. It would be a direct conclusion that the reduction of the packet loss is due to the packet retransmission, but then the percentages of the previous packet loss and the current retransmission would be similar, which does not happen. It is also thanks to the rest of parameters of the protocol, such as the beacons. From another point of view, the time between sent packets does

not correspond to the sampling time that was implemented to the motes. This happens because the timer functions integrated in TinyOs work with binary units. Then, the 10000ms periodic functions implemented, regarding that 1s correspond to 1024ms, have a real period of 9.765625 seconds. With this correction, the values of mean time between packets are really precise. It must be the half of this for the powerplugs, because they send two packets every 10s, and 8.57s for the temperature and light sensors, because they send 7 packets per minute. Finally, about the intermediate mote, the mean time should be 882.35ms if we consider the 68 packets it sends per minute. All these mean numbers may be decreased more or less significantly depending on the amount of retransmissions that the mote has made.

4.3 User profiling

4.3.1 Event classification

The method to identify the events for each sensor is explained in section 3.4. Figures 4.11 and 4.12 show the binary values when someone took a coffee and used water and when first a tea and later a coffee with milk are taken respectively. And figure 4.13 shows the output of matlab of the events occurred for someone having lunch that he had warmed in the microwave. It is possible to see an error on the log, it detects a fridge opening twice, and it should have detected the second one as a closing of the door. The way people open or close the door of the fridge is very subjective, so it is not possible to get a 100% accuracy about it.

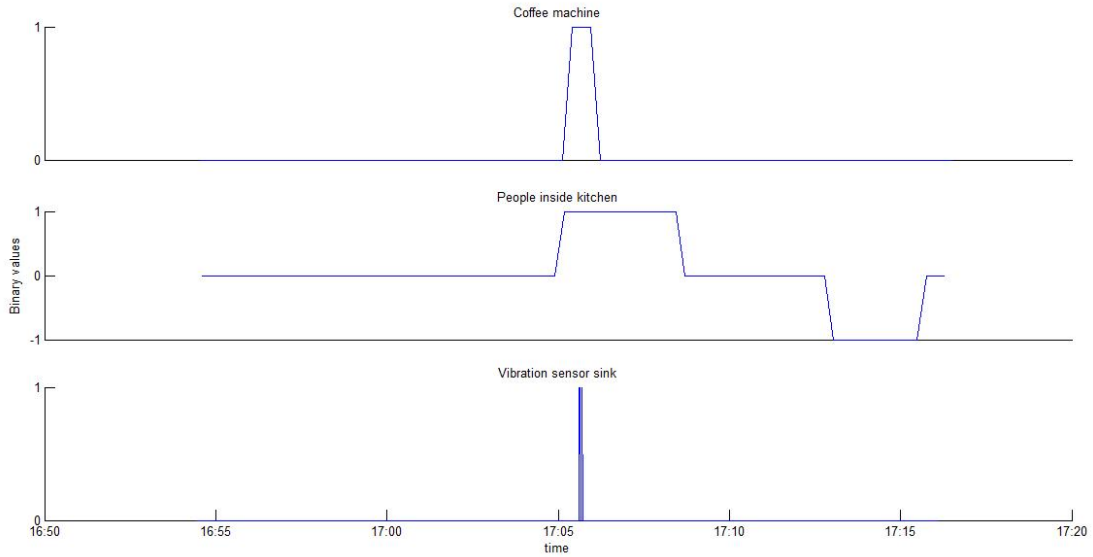


Figure 4.11: Binary values for the events of taking a coffee and using water

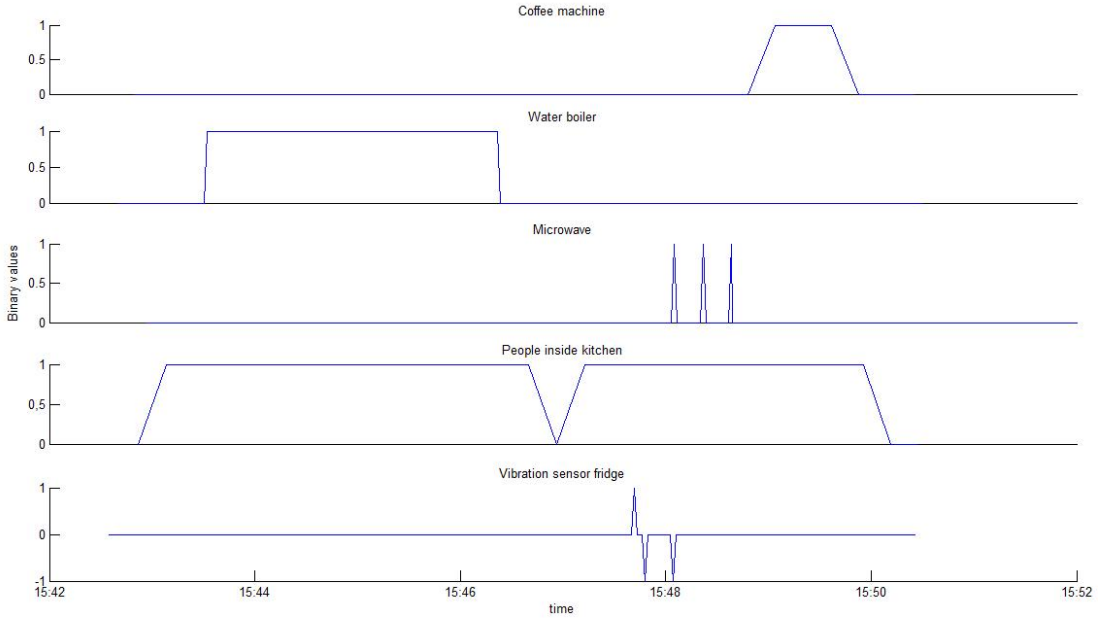


Figure 4.12: Binary values for the events of taking a tea and a coffee with milk

```
At 12:8 entered
At 12:8 entered
At 12:8 fridge opened
At 12:8 fridge opened
At 12:8 microwave started
At 12:9 microwave finished
At 12:10 sink
At 12:10 water
At 12:11 water
At 12:12 water
At 12:13 sink
At 12:13 left
At 12:13 someone warmed his food in the microwave and had lunch
```

Figure 4.13: Matlab output for someone having lunch warmed in the microwave

4.3.2 Pattern recognition

Finally, when the matrices with the binary values are already created and matlab has sent the output of the events, the big matrix with all events is generated. Then, matlab creates a vector containing the events and runs the algorithm of pattern detection. Figures 4.14 and 4.15 show the output of matlab and the vector of events for some actions registered. The meaning of the letters of the events vector are already explained in section 3.4.

```
E =

EdwLEwswswcweLELhdhcecweLwELewswwLEwLLbvwLE


At 16:17 someone entered, switched the dishwasher on and left
At 16:19 someone entered, washed his cup in the sink because the dishwasher was working
 and took a coffee
At 16:46 someone entered when the dishwasher had just finished and put his cup in the sink
 instead of emptying the dishwasher and putting it there
```

Figure 4.14: Matlab output for some actions

```
E =

LEEwwwwwcfeEwwswEcecfefgwLEEEEEcwwewcwwLewcEeLceEcececfgfeffcewwceL

At 14:55 someone entered, took a coffee with milk and left
At 15:1 people came for fika
At 15:2 someone entered, took a coffee and left
At 15:7 someone entered, took a coffee with milk and left
```

Figure 4.15: Matlab output for some actions

It is possible to see that there are many events registered, but as far as the analysis is hard to do when there is a lot of people on the room, only some patterns are identified. In figure 4.14, there is not a lot of people in the kitchen, so with just a few actions matlab already identifies some patterns. On the other hand, in figure 4.15, the kitchen is much more crowded, so even that there are much more events happening, it identifies the same number of patterns.

Chapter 5

Conclusion and Future work

5.1 Conclusion

The objectives of deploying an heterogeneous sensor network and gather data from all of them were well achieved. Even though, there is still a lot of work left to improve the system and provide more precise data. The wireless system has the huge benefit of avoiding the installation of wires everywhere, but on the contrary, it has many drawbacks too. The communication protocols have been improved significantly to guarantee a good number of data registered, but the communication with wires is, right now, much more faster and reliable.

Anyway, the recollection of data was performed efficciently, and the events and patterns identification worked properly too. Even though, there is also a big imprecision on the event detection, many of them depending on the person who uses the devices.

On another issue, the experiment has been a success in environmental points. The targets of collecting data about expenditures in water and electricity and detect non-environmentally friendly actions were properly completed.

Apart from this, there is also a privacy issue concerning all these work. Wireless networks can also be accessed from the outside of the building, so other people can collect data about the consumption in the building and people behaviour, it can easily be identified the number of people working there or any other parameters. Are we willing to give away some of our privacy in order to reduce our expenditures and help the world resources to survive? Or on the contrary, we are not going to allow other people introduce in our network, so the wireless networks have to be improved

to be safe to be ready to deploy in large scale?

5.2 Future work

The next steps about this research may differ depending on the point of view. For the data collection, much more sensors must be deployed everywhere around in order to gather data of absolutely everything. On the contrary, this will complicate a good communication, so then it might be better to deploy the least sensors possible gathering most of the data, so the communication is not so dense and it is more reliable.

Another issue would be to interact with the environment, not just collecting data, but also increasing or decreasing the heating, or turning on and off lights and devices when they are not being used.

Also, as stated before, there is a privacy concern that must be solved introducing security systems in the wireless network, such as an encryption of the signal, but this would also slow down the processing of data.

Bibliography

[1] European Environment Agency "Use of freshwater resources", 2009 [online]. Available: $http://www.eea.europa.eu/data-and-maps/indicators/use-of-freshwater-resources$

[2] European Environment Agency "EN18 Electricity Consumption", 2008 [online]. Available: $http://www.eea.europa.eu/data-and-maps/indicators/en18-electricity-consumption-1$

[3] Anon. wikipedia, 2010 [online]. Available: $http://en.wikipedia.org/wiki/Carrier\_sense\_multiple\_access$

[4] Anon. wikipedia, 2010 [online]. Available: $http://en.wikipedia.org/wiki/Time\_division\_multiple\_access$

[5] IEEE Std 802.15.4, 2007 [online]. Available: $http://voiplab.niu.edu.tw/IEEE/802.15/802.15.4a-2007.pdf$

[6] Younghun Kim, Thomas Schmid, Mani B. Srivastava, Yan Wang "Challenges in Resource Monitoring for Residential Spaces", University of California, Los Angeles, 2009

[7] Xiaofan Jiang, Minh Van Ly, Jay Taneja, Prabal Dutta, David Culler "Experiences with a High-Fidelity Wireless Building Energy Auditing Network", University of California, Berkeley, 2009

[8] Marshini Chetty, David Tran, Rebecca E. Grinter "Getting to Green: Understanding Resource Consumption in the Home", Georgia Institute of Technology, 2008

[9] Emmanuel Munguia Tapia "Activity Recognition in the Home Setting Using Simple and Ubiquitous Sensors", MSc Thesis, MIT, 2003

[10] J.P. Zimmermann, "End-use metering campaign in 400 households In Sweden", Department of System Analysis, Energimyndigheten, 2009

[11] Tmote Sky Data Sheet, Moteiv, Sanfrancisco, CA, 2006. [Online]. Available: *http* : *//www.moteiv.com/products/docs/tmote − skydatasheet.pdf*

[12] The tinyos community forum. UC Berkeley, 2004 [online]. Available: *http* : *//www.tinyos.net*

[13] Mealy, George H. "A Method to Synthesizing Sequential Circuits,. Bell Systems Technical Journal, pp. 1045–1079, 1955

[14] Photodiode S1087 Data Sheet, Hamamatsu, Japan, 2006 [online]. Available: *http* : *//sales.hamamatsu.com/assets/pdf/parts*$_S$*/S*1087$_e$*tc.pdf*

[15] Humidity and Temperature sensor SHT11 Data Sheet, Sensirion, 4Switzerland, 2009 [online]. Available: *http* : *//www.sensirion.com/en/pdf/product_information/Datasheet − humidity − sensor − SHT*1*x.pdf*

[16] 1104 - Vibration Sensor Data Sheet, Phidgets, [online]. Available: *http* : *//www.phidgets.com/documentation/Phidgets/*1104*.pdf*

[17] GP2Y0A21YK0F InfraRed Sensor Data Sheet, Sharp, 2006 [online]. Available: *http* : *//www.phidgets.com/documentation/Phidgets/*3521*Datasheet.pdf*

[18] Kui Liu, José Araújo, Henrik Sandberg and Karl Henrik Johansson, "SmartOffice: Intelligent Plug Design with Light Control Applications", MSc Thesis, KTH Stockholm, 2010

[19] CC2420 Chipcon Data Sheet, Chipcon, 2004 [online]. Available: *http* : *//www − inst.eecs.berkeley.edu/ cs*150*/Documents/CC*2420*.pdf*

[20] Aitor Hernández "Wireless Process Control using IEEE 802.15.4 Protocol" MSc Thesis, KTH Stockholm, 2010

[21] U. Tiberi, C. Fischione, K.H. Johansson and M.D. Di Benedetto "Adaptive Self-triggered Control over IEEE 802.15.4 Networks", KTH Stockholm, 2010

[22] CC2420DK Development Kit Data Sheet, Texas Instrument, [online]. Available: *http* : *//focus.ti.com/lit/ug/swru*045*/swru*045*.pdf*

[23] SmartRF^{TM} Packet Sniffer User Manual , Texas Instrument, [online]. Available: *http* : *//focus.ti.com/lit/ug/swru*187*b/swru*187*b.pdf*

Appendix A

Code for each one of the sensors

All codes used to program each one of the motes with its sensor are shown here. These are the final versions of the code with the IEEE 802.15.4 Standard communication implemented.

app_profile.h: file included in all motes

```
#ifndef __APP_PROFILE_H
#define __APP_PROFILE_H

enum {
  RADIO_CHANNEL = 25,
  PAN_ID = 0x1234,
  BASE_NODEID=10,
  INTERMEDIATE_NODEID=11,
  COORDINATOR_ADDRESS = INTERMEDIATE_NODEID,
  BEACON_ORDER = 6,
  SUPERFRAME_ORDER = 6,
  TX_POWER = 0, // in dBm
  AM_MYMSG = 10, //Radio channel number
  BUFFER_SIZE=10,
  FIRST_PLUG=1,
  SECOND_PLUG=2,
};

//structure of message
typedef nx_struct MyMsg{// max 28 bytes
   nx_uint8_t other;
   nx_uint8_t srcId;
   nx_uint16_t trgtId;
   nx_uint16_t data[BUFFER_SIZE];
}MyMsg;

#endif
```

Figure A.1: app_profile.h

Makefile: file to compile the program. It's the same for all motes except changing the name of the program in the first line of the code.

```
COMPONENT=CMPowerPlugAppC //change this depending on what to compile
CFLAGS += -I$(shell pwd)/../..
CFLAGS += -I$(shell pwd)/..
PFLAGS += -DIEEE154_BEACON_TX_DISABLED -DTKN154_DEBUG
#enable print function
CFLAGS += -I$(TOSDIR)/lib/printf
PFLAGS += -DPRINTF_BUFFER_SIZE=1000

include ../../Makefile.include
```

Figure A.2: Makefile

CMPowerPlugAppC.nc: code for the powerplug connected to the coffee machine. The code for the other powerplug is almost the same.

```
#include "Msp430Adc12.h"
#include <Timer.h>
#include "printf.h"
#include "PowerPlug.h"

#include "app_profile.h"

configuration CMPowerPlugAppC {
}
implementation
{
  //components implemented
  components MainC,
    new TimerMilliC() as Timer0,
    new TimerMilliC() as Timer1,
    new TimerMilliC() as Timer3,
    LedsC,
    HplMsp430GeneralIOC,
    Ieee802154BeaconEnabledC as MAC;

  components CMPowerPlugC,
             new Msp430Adc12ClientAutoRVGC() as AutoAdc;

  CMPowerPlugC -> MainC.Boot;
  CMPowerPlugC.Leds -> LedsC;
  CMPowerPlugC.Resource -> AutoAdc;
  AutoAdc.AdcConfigure -> CMPowerPlugC;
  CMPowerPlugC.Timer0 -> Timer0;
  CMPowerPlugC.Timer1 -> Timer1;
  CMPowerPlugC.Timer3 -> Timer3;

  CMPowerPlugC.MultiChannel -> AutoAdc.Msp430Adc12MultiChannel;

  CMPowerPlugC.MLME_SCAN -> MAC;
  CMPowerPlugC.MLME_SYNC -> MAC;
  CMPowerPlugC.MLME_BEACON_NOTIFY -> MAC;
  CMPowerPlugC.MLME_SYNC_LOSS -> MAC;
  CMPowerPlugC.MCPS_DATA -> MAC;
  CMPowerPlugC.Frame -> MAC;
  CMPowerPlugC.BeaconFrame -> MAC;
  CMPowerPlugC.Packet -> MAC;

  CMPowerPlugC.MLME_RESET -> MAC;
  CMPowerPlugC.MLME_SET -> MAC;
  CMPowerPlugC.MLME_GET -> MAC;

}
```

Figure A.3: CMPowerPlugAppC.nc

CMPowerPlugC.nc: code for the powerplug connected to the coffee machine

```
#include <Timer.h>
#include "printf.h"
#include "PowerPlug.h"

#include "TKN154.h"
#include "app_profile.h"

module CMPowerPlugC
{
  uses interface Boot;
  uses interface Resource;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer3;
  uses interface Msp430Adc12MultiChannel as MultiChannel;
  provides interface AdcConfigure<const msp430adc12_channel_config_t*>;

  uses {
    interface MCPS_DATA;
    interface MLME_RESET;
    interface MLME_SET;
    interface MLME_GET;
    interface MLME_SCAN;
    interface MLME_SYNC;
    interface MLME_BEACON_NOTIFY;
    interface MLME_SYNC_LOSS;
    interface IEEE154Frame as Frame;
    interface IEEE154BeaconFrame as BeaconFrame;
    interface Packet;
  }
}

implementation
{
//adc channel configuration
  const msp430adc12_channel_config_t config = {
    INPUT_CHANNEL_A5, REFERENCE_VREFplus_AVss, REFVOLT_LEVEL_2_5,
    SHT_SOURCE_SMCLK, SHT_CLOCK_DIV_1, SAMPLE_HOLD_64_CYCLES,
    SAMPCON_SOURCE_SMCLK, SAMPCON_CLOCK_DIV_1
};

//define variations or constants
  uint16_t *data_ptr;
  uint16_t buffer[2*BUFFER_SIZE];
  uint16_t Plug1;
  uint16_t Plug2[BUFFER_SIZE];
  uint16_t Ts;
  uint8_t j;
  uint8_t i;

  MyMsg* datapkt;
  uint8_t m_payloadLen = sizeof(MyMsg);
  message_t m_frame;
  ieee154_PANDescriptor_t m_PANDescriptor;
  bool m_ledCount;
  bool m_wasScanSuccessful;

  void startApp();
  void task getData();
  task void sendMessage();
  task void sendMessagePlug2();
```

Figure A.4: CMPowerPlugC.nc

```
//boot start
  event void Boot.booted()
  {
    datapkt =(MyMsg*)(call Packet.getPayload(&m_frame,sizeof(MyMsg)));
    datapkt->srcId = TOS_NODE_ID;
    Ts=10000;
    i=0;
    call MLME_RESET.request(TRUE);
  }

/*****************************************************************************
@name:      sendMessage

@function:  send the nodeId and current value to receiver(base station)

@parameter: pointer to payload

@return:    none
*****************************************************************************/

 task void sendMessage(){
      if (!m_wasScanSuccessful){
        // call Leds.led0Toggle();
        return;
      }
      else {

         //atomic{

          datapkt->data[0] = Plug1;
          for (j=1;j<BUFFER_SIZE;j++){
                      datapkt->data[j]=0;
                }

          datapkt->trgtId = INTERMEDIATE_NODEID;
          datapkt->other = FIRST_PLUG;
         }

       if (call MCPS_DATA.request  (
          &m_frame,                         // frame,
          m_payloadLen,                     // payloadLength,
          0,                                // msduHandle,
          TX_OPTIONS_ACK // TxOptions,
          ) != IEEE154_SUCCESS)
      call Leds.led0Toggle(); //fail!
      else
      call Leds.led0Off();

       //send current info from another plug


       Plug1=0;
       call Timer1.startOneShot(200);
    }

    task void sendMessagePlug2(){
      if (!m_wasScanSuccessful){
        // call Leds.led0Toggle();
        return;
      }
      else {
         atomic{
          for (j=0;j<BUFFER_SIZE;j++){
                      datapkt->data[j]=Plug2[j];
                }
```

Figure A.5: CMPowerPlugC.nc

```
          datapkt->trgtId = INTERMEDIATE_NODEID;
          datapkt->other = SECOND_PLUG;
         }

         if (call MCPS_DATA.request  (
          &m_frame,                        // frame,
          m_payloadLen,                    // payloadLength,
          0,                               // msduHandle,
          TX_OPTIONS_ACK // TxOptions,
          ) != IEEE154_SUCCESS)
      call Leds.led0Toggle(); //fail!
      else
      call Leds.led0Off();
      }
      j=0;
      i=0;
    }

 async command const msp430adc12_channel_config_t* AdcConfigure.getConfiguration()
  {
    return &config;
  }

/************************************************************************
@name:      getData

@function:  request the data from ADC readings

@parameter: none

@return:    none
************************************************************************/

  void task getData()
  {
    if (!call Resource.isOwner()){
         call Resource.request();
    }
    else {
         i=0;
            j=0;
            call Timer0.startPeriodic(100);
            call Timer3.startPeriodic(Ts);
    }
  }

  event void Resource.granted()
  {
    adc12memctl_t memctl[] ={{INPUT_CHANNEL_A0, REFERENCE_VREFplus_AVss},{INPUT_CHANNEL_A1, REFERENCE_VREFplus_AVss}};
    if (call MultiChannel.configure(&config, memctl, 2, buffer, 18, 500) == SUCCESS){
            i=0;
            j=0;
            call Timer0.startPeriodic(100);
            call Timer3.startPeriodic(Ts);
            }
  }

//timer fired, get data
  event void Timer0.fired(){
      j=j+1;
      call MultiChannel.getData();
  }

//timer fired, send second plug data
  event void Timer1.fired(){
      post sendMessagePlug2();
  }
```

Figure A.6: CMPowerPlugC.nc

```
event void Timer3.fired(){
      post sendMessage();
  }

//when data is ready         call MLME_SET.macRxOnWhenIdle(TRUE);
  async event void MultiChannel.dataReady(uint16_t *buf, uint16_t numSamples)
  {
     //atomic{
      data_ptr = buf;

        Plug1=Plug1+(data_ptr[5]+data_ptr[8]+data_ptr[11]+data_ptr[14])/4;
        if (j>9){
           Plug2[i]=(data_ptr[4]+data_ptr[7]+data_ptr[10]+data_ptr[13])/4;
           i=i+1;
           j=0;
        }
  }


  /*****************************************************
  * IEEE 802.15.4
  *****************************************************/

  void startApp()
  {
    ieee154_phyChannelsSupported_t channelMask;
    uint8_t scanDuration = BEACON_ORDER;

    call MLME_SET.phyTransmitPower(TX_POWER);
    call MLME_SET.macShortAddress(TOS_NODE_ID);

    // scan only the channel where we expect the coordinator
    channelMask = ((uint32_t) 1) << RADIO_CHANNEL;

    // we want all received beacons to be signalled
    // through the MLME_BEACON_NOTIFY interface, i.e.
    // we set the macAutoRequest attribute to FALSE
    call MLME_SET.macAutoRequest(FALSE);
    m_wasScanSuccessful = FALSE;
    call MLME_SCAN.request  (
                            PASSIVE_SCAN,           // ScanType
                            channelMask,            // ScanChannels
                            scanDuration,           // ScanDuration
                            0x00,                   // ChannelPage
                            0,                      // EnergyDetectListNumEntries
                            NULL,                   // EnergyDetectList
                            0,                      // PANDescriptorListNumEntries
                            NULL,                   // PANDescriptorList
                            0                       // security
                          );
  }

  event void MLME_RESET.confirm(ieee154_status_t status)
  {
    if (status == IEEE154_SUCCESS)
      startApp();
  }

  event message_t* MLME_BEACON_NOTIFY.indication (message_t* frame)
  {
    // received a beacon frame
    ieee154_phyCurrentPage_t page = call MLME_GET.phyCurrentPage();
    ieee154_macBSN_t beaconSequenceNumber = call BeaconFrame.getBSN(frame);
```

Figure A.7: CMPowerPlugC.nc

```
if (!m_wasScanSuccessful) {
      // received a beacon during channel scanning
      if (call BeaconFrame.parsePANDescriptor(
            frame, RADIO_CHANNEL, page, &m_PANDescriptor) == SUCCESS) {
        // let's see if the beacon is from our coordinator...
        if (m_PANDescriptor.CoordAddrMode == ADDR_MODE_SHORT_ADDRESS &&
            m_PANDescriptor.CoordPANId == PAN_ID &&
            m_PANDescriptor.CoordAddress.shortAddress == COORDINATOR_ADDRESS){
          // yes! wait until SCAN is finished, then syncronize to the beacons
          m_wasScanSuccessful = TRUE;
        }
      }
    } else {
      // received a beacon during synchronization, toggle LED2
      if (beaconSequenceNumber & 1)
        call Leds.led2On();
      else
        call Leds.led2Off();
    }

    return frame;
  }

  event void MLME_SCAN.confirm    (
                          ieee154_status_t status,
                          uint8_t ScanType,
                          uint8_t ChannelPage,
                          uint32_t UnscannedChannels,
                          uint8_t EnergyDetectListNumEntries,
                          int8_t* EnergyDetectList,
                          uint8_t PANDescriptorListNumEntries,
                          ieee154_PANDescriptor_t* PANDescriptorList
                        )
  {
    if (m_wasScanSuccessful) {
      // we received a beacon from the coordinator before
      call MLME_SET.macCoordShortAddress(m_PANDescriptor.CoordAddress.shortAddress);
      call MLME_SET.macPANId(m_PANDescriptor.CoordPANId);
      call MLME_SYNC.request(m_PANDescriptor.LogicalChannel, m_PANDescriptor.ChannelPage, TRUE);
      call MLME_SET.macRxOnWhenIdle(TRUE);
      call Frame.setAddressingFields(
          &m_frame,
          ADDR_MODE_SHORT_ADDRESS,        // SrcAddrMode,
          ADDR_MODE_SHORT_ADDRESS,        // DstAddrMode,
          m_PANDescriptor.CoordPANId,     // DstPANId,
          &m_PANDescriptor.CoordAddress,  // DstAddr,
          NULL                            // security
          );
          post getData();
    } else
      startApp();
  }

  event void MCPS_DATA.confirm    (
                          message_t *msg,
                          uint8_t msduHandle,
                          ieee154_status_t status,
                          uint32_t timestamp
                        )
  {
    if (status == IEEE154_SUCCESS && m_ledCount++ >= 2) {
      m_ledCount = 0;
      call Leds.led1Toggle();
    }
  // previous
   //post packetSendTask();
  }
```

Figure A.8: CMPowerPlugC.nc

```
event void MLME_SYNC_LOSS.indication(
                          ieee154_status_t lossReason,
                          uint16_t PANId,
                          uint8_t LogicalChannel,
                          uint8_t ChannelPage,
                          ieee154_security_t *security)
  {
    m_wasScanSuccessful = FALSE;
    call Leds.led1Off();
    call Leds.led2Off();
    call Timer0.stop();
    call Timer1.stop();
    call Timer3.stop();

    startApp();
  }

  event message_t* MCPS_DATA.indication (message_t* frame)
   {
     if (call Frame.getPayloadLength(frame) == m_payloadLen &&
                                      call Frame.getFrameType(frame) == FRAMETYPE_DATA) {
     MyMsg *receivepkt = (MyMsg*) (call Packet.getPayload(frame,m_payloadLen ));

     if(receivepkt->srcId==BASE_NODEID){

        if(receivepkt->other==3 && receivepkt->trgtId==111){
           call Leds.led0Toggle();
           call Timer0.stop();
           call Timer3.stop();
        }
        else if(receivepkt->other==4 && receivepkt->trgtId==TOS_NODE_ID){
           Ts=receivepkt->data[0];
        }
       }
     }
     return frame;
   }
}//end of implementation
```

Figure A.9: CMPowerPlugC.nc

LightTempSensorAppC.nc: code for the temperature and light sensors mote

```
#include "Msp430Adc12.h"
#include <Timer.h>
#include "printf.h"

#include "app_profile.h"

configuration LightTempSensorAppC {
}
implementation
{
  components MainC,
    new TimerMilliC() as Timer1,
    new TimerMilliC() as Timer2,
    LightTempSensorC,
    LedsC,
    new  SensirionSht11C() as Sensirion,
    Ieee802154BeaconEnabledC as MAC,
    new HamamatsuS10871TsrC() as LightSensor;// component used to measure the light intensity

  //connections
  LightTempSensorC -> MainC.Boot;
  LightTempSensorC.Leds -> LedsC;
  LightTempSensorC.Timer1 -> Timer1;
  LightTempSensorC.Timer2 -> Timer2;
  LightTempSensorC.Temperature -> Sensirion.Temperature;
  LightTempSensorC.ReadLight->LightSensor;

  LightTempSensorC.MLME_SCAN -> MAC;
  LightTempSensorC.MLME_SYNC -> MAC;
  LightTempSensorC.MLME_BEACON_NOTIFY -> MAC;
  LightTempSensorC.MLME_SYNC_LOSS -> MAC;
  LightTempSensorC.MCPS_DATA -> MAC;
  LightTempSensorC.Frame -> MAC;
  LightTempSensorC.BeaconFrame -> MAC;
  LightTempSensorC.Packet -> MAC;


  LightTempSensorC.MLME_RESET -> MAC;
  LightTempSensorC.MLME_SET -> MAC;
  LightTempSensorC.MLME_GET -> MAC;

}
```

Figure A.10: LightTempSensorAppC.nc

LightTempSensorC.nc: code for the temperature and light sensors mote

```
#include <Timer.h>
#include "printf.h"
#include "TemperatureSensor.h"

#include "TKN154.h"
#include "app_profile.h"

module LightTempSensorC
{
  uses interface Boot;
  uses interface Leds;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Read<uint16_t> as Temperature;
  uses interface Read<uint16_t> as ReadLight;
  uses interface HalSht11Advanced;
  uses {
    interface MCPS_DATA;
    interface MLME_RESET;
    interface MLME_SET;
    interface MLME_GET;
    interface MLME_SCAN;
    interface MLME_SYNC;
    interface MLME_BEACON_NOTIFY;
    interface MLME_SYNC_LOSS;
    interface IEEE154Frame as Frame;
    interface IEEE154BeaconFrame as BeaconFrame;
    interface Packet;
  }
}

implementation
{
  bool busy=FALSE;  //whether radio is busy or not
  uint16_t temperature;
  uint16_t intensity;
  uint16_t TsT;
  uint16_t TsL;
  uint8_t j;

  MyMsg* datapkt;
  uint8_t m_payloadLen = sizeof(MyMsg);
  message_t m_frame;
  ieee154_PANDescriptor_t m_PANDescriptor;
  bool m_ledCount;
  bool m_wasScanSuccessful;

  void startApp();
  task void sendMessageTemp();
  task void sendMessageLight();

  //boot start
  event void Boot.booted()
  {
      datapkt =(MyMsg*)(call Packet.getPayload(&m_frame,sizeof(MyMsg)));
      datapkt->srcId = TOS_NODE_ID;
      TsT=60000;
      TsL=10000;
      call MLME_RESET.request(TRUE);
  }
```

Figure A.11: LightTempSensorC.nc

```
event void Timer1.fired()
{
    call Temperature.read();
}

event void Timer2.fired()
{
    call ReadLight.read();
}

//when readtemp request has been processed
event void Temperature.readDone(error_t error, uint16_t value)
{
    temperature=value;
    post sendMessageTemp();
}

event void ReadLight.readDone(error_t error, uint16_t value)
{
    intensity=value;
    post sendMessageLight();
}

task void sendMessageLight(){

    if (!m_wasScanSuccessful){
      // call Leds.led0Toggle();
      return;
    }
    else {
        atomic{
        datapkt->data[0] = intensity;
        for (j=1;j<10;j++){
           datapkt->data[j]=0;
        }
        datapkt->trgtId = INTERMEDIATE_NODEID;
        datapkt->other = 5;
      }
       if (call MCPS_DATA.request  (
        &m_frame,                       // frame,
        m_payloadLen,                   // payloadLength,
        0,                              // msduHandle,
        TX_OPTIONS_ACK // TxOptions,
        ) != IEEE154_SUCCESS)
    call Leds.led0Toggle(); //fail!
    else
    call Leds.led0Off();
    }
}

//send the light intensity to the base station
task void sendMessageTemp(){
    if (!m_wasScanSuccessful){
      // call Leds.led0Toggle();
      return;
    }
    else {
       atomic{
        datapkt->data[0] = temperature;
        for (j=1;j<10;j++){
           datapkt->data[j]=0;
        }
        datapkt->trgtId = INTERMEDIATE_NODEID;
        datapkt->other = 6;
       }
```

Figure A.12: LightTempSensorC.nc

```
  if (call MCPS_DATA.request  (
          &m_frame,                       // frame,
          m_payloadLen,                   // payloadLength,
          0,                              // msduHandle,
          TX_OPTIONS_ACK // TxOptions,
          ) != IEEE154_SUCCESS)
      call Leds.led0Toggle(); //fail!
      else
      call Leds.led0Off();

      }
  }

  event void HalSht11Advanced.getVoltageStatusDone(error_t error, bool isLow) {}

  event void HalSht11Advanced.setHeaterDone(error_t error) {}

  event void HalSht11Advanced.setResolutionDone(error_t error) {}

/*****************************************************
  * IEEE 802.15.4
  *****************************************************/

  void startApp()
  {
    ieee154_phyChannelsSupported_t channelMask;
    uint8_t scanDuration = BEACON_ORDER;

    call MLME_SET.phyTransmitPower(TX_POWER);
    call MLME_SET.macShortAddress(TOS_NODE_ID);

    // scan only the channel where we expect the coordinator
    channelMask = ((uint32_t) 1) << RADIO_CHANNEL;

    // we want all received beacons to be signalled
    // through the MLME_BEACON_NOTIFY interface, i.e.
    // we set the macAutoRequest attribute to FALSE
    call MLME_SET.macAutoRequest(FALSE);
    m_wasScanSuccessful = FALSE;
    call MLME_SCAN.request  (
                            PASSIVE_SCAN,           // ScanType
                            channelMask,            // ScanChannels
                            scanDuration,           // ScanDuration
                            0x00,                   // ChannelPage
                            0,                      // EnergyDetectListNumEntries
                            NULL,                   // EnergyDetectList
                            0,                      // PANDescriptorListNumEntries
                            NULL,                   // PANDescriptorList
                            0                       // security
                          );
  }

  event void MLME_RESET.confirm(ieee154_status_t status)
  {
    if (status == IEEE154_SUCCESS)
      startApp();
  }

  event message_t* MLME_BEACON_NOTIFY.indication (message_t* frame)
  {
    // received a beacon frame
    ieee154_phyCurrentPage_t page = call MLME_GET.phyCurrentPage();
    ieee154_macBSN_t beaconSequenceNumber = call BeaconFrame.getBSN(frame);
```

Figure A.13: LightTempSensorC.nc

```
if (!m_wasScanSuccessful) {
    // received a beacon during channel scanning
    if (call BeaconFrame.parsePANDescriptor(
          frame, RADIO_CHANNEL, page, &m_PANDescriptor) == SUCCESS) {
      // let's see if the beacon is from our coordinator...
      if (m_PANDescriptor.CoordAddrMode == ADDR_MODE_SHORT_ADDRESS &&
          m_PANDescriptor.CoordPANId == PAN_ID &&
          m_PANDescriptor.CoordAddress.shortAddress == COORDINATOR_ADDRESS){
        // yes! wait until SCAN is finished, then syncronize to the beacons
        m_wasScanSuccessful = TRUE;
      }
    }
  } else {
    // received a beacon during synchronization, toggle LED2
    if (beaconSequenceNumber & 1)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }
  return frame;
}

event void MLME_SCAN.confirm    (
                        ieee154_status_t status,
                        uint8_t ScanType,
                        uint8_t ChannelPage,
                        uint32_t UnscannedChannels,
                        uint8_t EnergyDetectListNumEntries,
                        int8_t* EnergyDetectList,
                        uint8_t PANDescriptorListNumEntries,
                        ieee154_PANDescriptor_t* PANDescriptorList
                      )
{
  if (m_wasScanSuccessful) {
    // we received a beacon from the coordinator before
    call MLME_SET.macCoordShortAddress(m_PANDescriptor.CoordAddress.shortAddress);
    call MLME_SET.macPANId(m_PANDescriptor.CoordPANId);
    call MLME_SYNC.request(m_PANDescriptor.LogicalChannel, m_PANDescriptor.ChannelPage, TRUE);
    call MLME_SET.macRxOnWhenIdle(TRUE);
    call Frame.setAddressingFields(
        &m_frame,
        ADDR_MODE_SHORT_ADDRESS,        // SrcAddrMode,
        ADDR_MODE_SHORT_ADDRESS,        // DstAddrMode,
        m_PANDescriptor.CoordPANId,     // DstPANId,
        &m_PANDescriptor.CoordAddress,  // DstAddr,
        NULL                            // security
        );
        call Timer1.startPeriodicAt(1300,TsT);
        call Timer2.startPeriodicAt(1200,TsL);
  } else
    startApp();
}

event void MCPS_DATA.confirm    (
                        message_t *msg,
                        uint8_t msduHandle,
                        ieee154_status_t status,
                        uint32_t timestamp
                      )
{
  if (status == IEEE154_SUCCESS && m_ledCount++ >= 2) {
    m_ledCount = 0;
    call Leds.led1Toggle();
  }
// previous
 //post packetSendTask();
}
```

Figure A.14: LightTempSensorC.nc

```
  event void MLME_SYNC_LOSS.indication(
                          ieee154_status_t lossReason,
                          uint16_t PANId,
                          uint8_t LogicalChannel,
                          uint8_t ChannelPage,
                          ieee154_security_t *security)
  {
    m_wasScanSuccessful = FALSE;
    call Leds.led1Off();
    call Leds.led2Off();
    call Timer1.stop();
    call Timer2.stop();
    startApp();
  }

  event message_t* MCPS_DATA.indication (message_t* frame)
   {
     if (call Frame.getPayloadLength(frame) == m_payloadLen &&
                                      call Frame.getFrameType(frame) == FRAMETYPE_DATA) {
     MyMsg *receivepkt = (MyMsg*) (call Packet.getPayload(frame,m_payloadLen ));

     if(receivepkt->srcId==BASE_NODEID){

        if(receivepkt->other==3 && receivepkt->trgtId==111){
           call Timer1.stop();
           call Timer2.stop();
        }
        else if(receivepkt->other==4 && receivepkt->trgtId==TOS_NODE_ID){
           TsL=receivepkt->data[0];
           TsT=receivepkt->data[0]*30;
        }
       }
     }
     return frame;
   }
}//end of implementation
```

Figure A.15: LightTempSensorC.nc

VibrationSensorAppC.nc: code for the vibration sensor mote

```
#include <Timer.h>
#include "printf.h"
#include "Vibration.h"
#include "Msp430Adc12.h"

#include "app_profile.h"

configuration VibrationSensorAppC {}
implementation {

  components VibrationSensorC, MainC, LedsC,     Ieee802154BeaconEnabledC as MAC;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;
  components TestPrintfC;
  components new VibrationC() as Sensor;

  VibrationSensorC.Boot -> MainC;
  VibrationSensorC.Leds -> LedsC;
  VibrationSensorC.Timer1 -> Timer1;
  VibrationSensorC.Timer2 -> Timer2;
  VibrationSensorC.Read -> Sensor;

  VibrationSensorC.MLME_SCAN -> MAC;
  VibrationSensorC.MLME_SYNC -> MAC;
  VibrationSensorC.MLME_BEACON_NOTIFY -> MAC;
  VibrationSensorC.MLME_SYNC_LOSS -> MAC;
  VibrationSensorC.MCPS_DATA -> MAC;
  VibrationSensorC.Frame -> MAC;
  VibrationSensorC.BeaconFrame -> MAC;
  VibrationSensorC.Packet -> MAC;

  VibrationSensorC.MLME_RESET -> MAC;
  VibrationSensorC.MLME_SET -> MAC;
  VibrationSensorC.MLME_GET -> MAC;
}
```

Figure A.16: VibrationSensorAppC.nc

VibrationSensorC.nc: code for the vibration sensor mote

```
#include "Timer.h"
#include "AM.h"
#include "printf.h"
#include "Vibration.h"

#include "TKN154.h"
#include "app_profile.h"

module VibrationSensorC
{
  uses {
    interface Boot;
    interface Leds;
    interface Timer<TMilli> as Timer1;
    interface Timer<TMilli> as Timer2;
    interface Read<uint16_t>;
  }
  uses {
    interface MCPS_DATA;
    interface MLME_RESET;
    interface MLME_SET;
    interface MLME_GET;
    interface MLME_SCAN;
    interface MLME_SYNC;
    interface MLME_BEACON_NOTIFY;
    interface MLME_SYNC_LOSS;
    interface IEEE154Frame as Frame;
    interface IEEE154BeaconFrame as BeaconFrame;
    interface Packet;
  }
}
implementation
{

  bool busy=FALSE;  //whether radio is busy or not
  uint16_t vibration[BUFFER_SIZE];
  uint16_t maxvalues[BUFFER_SIZE];
  uint16_t Ts;
  uint8_t i;
  uint8_t j;
  uint8_t k;

  MyMsg* datapkt;
  uint8_t m_payloadLen = sizeof(MyMsg);
  message_t m_frame;
  ieee154_PANDescriptor_t m_PANDescriptor;
  bool m_ledCount;
  bool m_wasScanSuccessful;
  task void sendMessage();

  void startApp();

  //boot start
  event void Boot.booted()
  {
    datapkt =(MyMsg*)(call Packet.getPayload(&m_frame,sizeof(MyMsg)));
    datapkt->srcId = TOS_NODE_ID;
    Ts=10000;
    i=0;
    k=0;
    call MLME_RESET.request(TRUE);
  }
```

Figure A.17: VibrationSensorC.nc

```
event void Timer1.fired()
{
  call Read.read();
}

event void Timer2.fired()
{
  post sendMessage();
  i=0;
}

event void Read.readDone(error_t result, uint16_t val){

  vibration[i]=val;
  i=i+1;
  if (i==10){
    for (j=0;j<BUFFER_SIZE;j++){
      if (maxvalues[k]>=vibration[j]){
        }
      else if (maxvalues[k]<vibration[j]){
        maxvalues[k]=vibration[j];
      }
    }
    k=k+1;
    i=0;
  }
}

//send the vibration intensity to the base station
task void sendMessage(){
    if (!m_wasScanSuccessful){
      // call Leds.led0Toggle();
      return;
    }
    else {
              for (j=0;j<BUFFER_SIZE;j++) {
           datapkt->data[j] = maxvalues[j];
        }
        datapkt->trgtId = INTERMEDIATE_NODEID;
        datapkt->other = 0;

      if (call MCPS_DATA.request  (
                &m_frame,                       // frame,
                m_payloadLen,                   // payloadLength,
                0,                              // msduHandle,
                TX_OPTIONS_ACK // TxOptions,
                ) != IEEE154_SUCCESS)
                call Leds.led0Toggle(); //fail!
              else
                 call Leds.led0Off();

              for (j=0;j<BUFFER_SIZE;j++){
                  vibration[j]=0;
                  maxvalues[j]=0;
              }
              k=0;
    }
}

/*****************************************************
* IEEE 802.15.4
*****************************************************/

void startApp()
{
  ieee154_phyChannelsSupported_t channelMask;
  uint8_t scanDuration = BEACON_ORDER;
```

Figure A.18: VibrationSensorC.nc

```
  call MLME_SET.phyTransmitPower(TX_POWER);
  call MLME_SET.macShortAddress(TOS_NODE_ID);
  // scan only the channel where we expect the coordinator
  channelMask = ((uint32_t) 1) << RADIO_CHANNEL;

  // we want all received beacons to be signalled
  // through the MLME_BEACON_NOTIFY interface, i.e.
  // we set the macAutoRequest attribute to FALSE
  call MLME_SET.macAutoRequest(FALSE);
  m_wasScanSuccessful = FALSE;
  call MLME_SCAN.request  (
                        PASSIVE_SCAN,           // ScanType
                        channelMask,            // ScanChannels
                        scanDuration,           // ScanDuration
                        0x00,                   // ChannelPage
                        0,                      // EnergyDetectListNumEntries
                        NULL,                   // EnergyDetectList
                        0,                      // PANDescriptorListNumEntries
                        NULL,                   // PANDescriptorList
                        0                       // security
                      );
}

event void MLME_RESET.confirm(ieee154_status_t status)
{
  if (status == IEEE154_SUCCESS)
    startApp();
}

event message_t* MLME_BEACON_NOTIFY.indication (message_t* frame)
{
  // received a beacon frame
  ieee154_phyCurrentPage_t page = call MLME_GET.phyCurrentPage();
  ieee154_macBSN_t beaconSequenceNumber = call BeaconFrame.getBSN(frame);
  if (!m_wasScanSuccessful) {
    // received a beacon during channel scanning
    if (call BeaconFrame.parsePANDescriptor(
          frame, RADIO_CHANNEL, page, &m_PANDescriptor) == SUCCESS) {
      // let's see if the beacon is from our coordinator...
      if (m_PANDescriptor.CoordAddrMode == ADDR_MODE_SHORT_ADDRESS &&
          m_PANDescriptor.CoordPANId == PAN_ID &&
          m_PANDescriptor.CoordAddress.shortAddress == COORDINATOR_ADDRESS){
        // yes! wait until SCAN is finished, then syncronize to the beacons
        m_wasScanSuccessful = TRUE;
      }
    }
  } else {
    // received a beacon during synchronization, toggle LED2
    if (beaconSequenceNumber & 1)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }
  return frame;
}

event void MLME_SCAN.confirm    (
                        ieee154_status_t status,
                        uint8_t ScanType,
                        uint8_t ChannelPage,
                        uint32_t UnscannedChannels,
                        uint8_t EnergyDetectListNumEntries,
                        int8_t* EnergyDetectList,
                        uint8_t PANDescriptorListNumEntries,
                        ieee154_PANDescriptor_t* PANDescriptorList
                      )
{
```

Figure A.19: VibrationSensorC.nc

```
  if (m_wasScanSuccessful) {
    // we received a beacon from the coordinator before
    call MLME_SET.macCoordShortAddress(m_PANDescriptor.CoordAddress.shortAddress);
    call MLME_SET.macPANId(m_PANDescriptor.CoordPANId);
    call MLME_SYNC.request(m_PANDescriptor.LogicalChannel, m_PANDescriptor.ChannelPage, TRUE);
    call MLME_SET.macRxOnWhenIdle(TRUE);
    call Frame.setAddressingFields(
        &m_frame,
        ADDR_MODE_SHORT_ADDRESS,        // SrcAddrMode,
        ADDR_MODE_SHORT_ADDRESS,        // DstAddrMode,
        m_PANDescriptor.CoordPANId,     // DstPANId,
        &m_PANDescriptor.CoordAddress,  // DstAddr,
        NULL                            // security
        );
        call Timer1.startPeriodicAt(2000,100);
        call Timer2.startPeriodicAt(2000,Ts);
  } else
    startApp();
}

event void MCPS_DATA.confirm    (
                        message_t *msg,
                        uint8_t msduHandle,
                        ieee154_status_t status,
                        uint32_t timestamp
                      )
{
  if (status == IEEE154_SUCCESS && m_ledCount++ >= 2) {
    m_ledCount = 0;
    call Leds.led1Toggle();
  }
}

event void MLME_SYNC_LOSS.indication(
                        ieee154_status_t lossReason,
                        uint16_t PANId,
                        uint8_t LogicalChannel,
                        uint8_t ChannelPage,
                        ieee154_security_t *security)
{
  m_wasScanSuccessful = FALSE;
  call Leds.led1Off();
  call Leds.led2Off();
  call Timer1.stop();
  call Timer2.stop();
  startApp();
}

event message_t* MCPS_DATA.indication (message_t* frame)
{
   call Leds.led0Toggle();
   if (call Frame.getPayloadLength(frame) == m_payloadLen && call Frame.getFrameType(frame) == FRAMETYPE_DATA) {
   MyMsg *receivepkt = (MyMsg*) (call Packet.getPayload(frame,m_payloadLen ));
   if(receivepkt->srcId==BASE_NODEID){
      if(receivepkt->other==3 && receivepkt->trgtId==111){
         call Timer1.stop();
         call Timer2.stop();
         i=0;
      }
      else if(receivepkt->other==4 && receivepkt->trgtId==TOS_NODE_ID){
         Ts=receivepkt->data[0];
      }
   }
   }
   return frame;
 }
}
```

Figure A.20: VibrationSensorC.nc

IRSensorAppC.nc: code for the InfraRed sensor mote

```
#include <Timer.h>
#include "printf.h"
#include "Msp430Adc12.h"

#include "app_profile.h"

configuration IR2AppC
{
}
implementation {

  components IR2C, MainC, LedsC, new TimerMilliC() as Timer0, new TimerMilliC() as Timer1,   Ieee802154BeaconEnabledC as MAC;
  components new Msp430Adc12ClientAutoRVGC() as AutoAdc;

  IR2C.Boot -> MainC;
  IR2C.Leds -> LedsC;
  IR2C.Resource -> AutoAdc;
  AutoAdc.AdcConfigure -> IR2C;
  IR2C.Timer0 -> Timer0;
  IR2C.Timer1 -> Timer1;

  IR2C.MultiChannel -> AutoAdc.Msp430Adc12MultiChannel;

  IR2C.MLME_SCAN -> MAC;
  IR2C.MLME_SYNC -> MAC;
  IR2C.MLME_BEACON_NOTIFY -> MAC;
  IR2C.MLME_SYNC_LOSS -> MAC;
  IR2C.MCPS_DATA -> MAC;
  IR2C.Frame -> MAC;
  IR2C.BeaconFrame -> MAC;
  IR2C.Packet -> MAC;

  IR2C.MLME_RESET -> MAC;
  IR2C.MLME_SET -> MAC;
  IR2C.MLME_GET -> MAC;

}
```

Figure A.21: IRSensorAppC.nc

IRSensorC.nc: code for the InfraRed sensor mote

```
#include "Timer.h"
#include "printf.h"

#include "TKN154.h"
#include "app_profile.h"

module IR2C
{
  uses {
    interface Boot;
    interface Leds;
    interface Timer<TMilli> as Timer0;
    interface Timer<TMilli> as Timer1;
  }
  uses interface Resource;
  uses interface Msp430Adc12MultiChannel as MultiChannel;
  provides interface AdcConfigure<const msp430adc12_channel_config_t*>;

  uses {
    interface MCPS_DATA;
    interface MLME_RESET;
    interface MLME_SET;
    interface MLME_GET;
    interface MLME_SCAN;
    interface MLME_SYNC;
    interface MLME_BEACON_NOTIFY;
    interface MLME_SYNC_LOSS;
    interface IEEE154Frame as Frame;
    interface IEEE154BeaconFrame as BeaconFrame;
    interface Packet;
  }
}
implementation
{
  //adc channel configuration
  const msp430adc12_channel_config_t config = {
    INPUT_CHANNEL_A5, REFERENCE_VREFplus_AVss, REFVOLT_LEVEL_1_5,
    SHT_SOURCE_SMCLK, SHT_CLOCK_DIV_1, SAMPLE_HOLD_4_CYCLES,
    SAMPCON_SOURCE_ACLK, SAMPCON_CLOCK_DIV_1
  };

  //define variations or constants
  uint16_t buffer[2*BUFFER_SIZE];
  uint16_t *data;
  uint16_t IR1;
  uint16_t IR2;
  uint8_t j;
  uint16_t people;
  uint8_t current_state;
  uint16_t Ts;
  bool busy=FALSE;  //show if the radio is used or not

  //variables used for the communication

  MyMsg* datapkt;
  uint8_t m_payloadLen = sizeof(MyMsg);
  message_t m_frame;
  ieee154_PANDescriptor_t m_PANDescriptor;
  bool m_ledCount;
  bool m_wasScanSuccessful;

  task void sendMessage();
```

Figure A.22: IRSensorC.nc

```
  void task getData();
  void task AnalyseData();
  void startApp();

  //boot start
  event void Boot.booted() {
    datapkt =(MyMsg*)(call Packet.getPayload(&m_frame,sizeof(MyMsg)));
    datapkt->srcId = TOS_NODE_ID;
    current_state=0;
    people=100;
    Ts=10000;
    call MLME_RESET.request(TRUE);
  }

  async command const msp430adc12_channel_config_t* AdcConfigure.getConfiguration()
  {
    return &config;
  }

  void task getData()
  {
if (!call Resource.isOwner()){
call Resource.request();
}
else {
call Timer0.startPeriodic(50);
}
  }

  task void sendMessage(){

      if (!m_wasScanSuccessful){
        //call Leds.led0Toggle();
        return;
      }
      else {
         atomic{
          datapkt->data[0] = people;
          for (j=1;j<10;j++) {
             datapkt->data[j]=0;
          }
          datapkt->trgtId=INTERMEDIATE_NODEID;
          datapkt->other = 7;
         }

        if (call MCPS_DATA.request  (
          &m_frame,                       // frame,
          m_payloadLen,                   // payloadLength,
          0,                              // msduHandle,
          TX_OPTIONS_ACK // TxOptions,
          ) != IEEE154_SUCCESS)
      call Leds.led0Toggle(); //fail!
      else
      call Leds.led0Off();
      }
  }

  void task AnalyseData()
  {

      IR1=(data[5]+data[8]+data[11]+data[14])/4;
      IR2=(data[4]+data[7]+data[10]+data[13])/4;

    if ( current_state==0) {
       if (IR1>1200 && IR2<1200) { //someone might be entering
         current_state=1;
       }
```

Figure A.23: IRSensorC.nc

```
    else if (IR1<1200 && IR2>1200) { //someone might be leaving
      current_state=2;
    }
    else if (IR1>1200 && IR2>1200) { //someone might be crossing
      current_state=3;
    }
}
else if ( current_state==1 ) {
  if ( IR1>1200 && IR2>1200){  //someone is crossing the door
    current_state=11;
  }
  else if( IR1>1200 && IR2<=1200) {  //is still at the beginning
  }
  else if( IR1<=1200 && IR2<=1200) {  //didnt enter
    current_state=0;
  }
  else if( IR1<1200 && IR2>1200) {  //crossing the door
    current_state=11;
  }
}
else if ( current_state==11 ) {
  if ( IR1>1200 && IR2>1200){  //someone is crossing the door
  }
  else if( IR1>1200 && IR2<=1200) { //came back
    current_state=1;
  }
  else if( IR1<=1200 && IR2<=1200) {  // entered
    people=people+1;
    printf("Someone entered the room\npeople:%u\n",people);
    printfflush();
    current_state=0;
  }
  else if( IR1<1200 && IR2>1200) {  //crossing the door
  }
}
else if ( current_state==2 ) {
  if ( IR1>1200 && IR2>1200){  //someone crossing the door
    current_state=21;
  }
  else if( IR1<=1200 && IR2<=1200) {  //didnt leave
    current_state=0;
  }
  else if( IR1<=1200 && IR2>1200) { //still at the beginning
  }
  else if( IR1>1200 && IR2<1200) {   // crossing the door
    current_state=21;
  }
}
else if ( current_state==21 ) {
  if ( IR1>1200 && IR2>1200){  //someone is crossing the door
  }
  else if( IR1<=1200 && IR2>1200) { //came back
    current_state=2;
  }
  else if( IR1<=1200 && IR2<=1200) {  // left
    people=people-1;
    printf("Someone left the room\npeople:%u\n",people);
    printfflush();
    if (people<0){
       people=0;
    }
    current_state=0;
  }
  else if( IR1>1200 && IR2<1200) {  //crossing the door
  }
}
```

Figure A.24: IRSensorC.nc

```
  else if ( current_state==3 ) {
    if (IR1<IR2) { //someone entering
      current_state=11;
    }
    else if (IR1>IR2) {    //someone leaving
      current_state=21;
    }
  }
  printf("state:%u\n",current_state);
  printfflush();
}


event void Resource.granted()
{
  adc12memctl_t memctl[] ={{INPUT_CHANNEL_A0, REFERENCE_VREFplus_AVss},{INPUT_CHANNEL_A1,REFERENCE_VREFplus_AVss}};
  if (call MultiChannel.configure(&config, memctl, 2, buffer, 18, 500) == SUCCESS){
         call Timer0.startPeriodic(50);
  }
}

event void Timer0.fired()
{
  call MultiChannel.getData();
}

event void Timer1.fired()
{
   post sendMessage();
}

//when data is ready
async event void MultiChannel.dataReady(uint16_t *buf, uint16_t numSamples)
{
   atomic{
      data = buf;
      post AnalyseData();
   }
}

/*****************************************************
* IEEE 802.15.4
*****************************************************/

void startApp()
{
  ieee154_phyChannelsSupported_t channelMask;
  uint8_t scanDuration = BEACON_ORDER;

  call MLME_SET.phyTransmitPower(TX_POWER);
  call MLME_SET.macShortAddress(TOS_NODE_ID);

  // scan only the channel where we expect the coordinator
  channelMask = ((uint32_t) 1) << RADIO_CHANNEL;

  // we want all received beacons to be signalled
  // through the MLME_BEACON_NOTIFY interface, i.e.
  // we set the macAutoRequest attribute to FALSE
  call MLME_SET.macAutoRequest(FALSE);
  m_wasScanSuccessful = FALSE;
  call MLME_SCAN.request  (
                        PASSIVE_SCAN,           // ScanType
                        channelMask,            // ScanChannels
                        scanDuration,           // ScanDuration
                        0x00,                   // ChannelPage
                        0,                      // EnergyDetectListNumEntries
```

Figure A.25: IRSensorC.nc

```
                        NULL,                   // EnergyDetectList
                        0,                      // PANDescriptorListNumEntries
                        NULL,                   // PANDescriptorList
                        0                       // security
                      );
}

event void MLME_RESET.confirm(ieee154_status_t status)
{
  if (status == IEEE154_SUCCESS)
    startApp();
}

event message_t* MLME_BEACON_NOTIFY.indication (message_t* frame)
{
  // received a beacon frame
  ieee154_phyCurrentPage_t page = call MLME_GET.phyCurrentPage();
  ieee154_macBSN_t beaconSequenceNumber = call BeaconFrame.getBSN(frame);

  if (!m_wasScanSuccessful) {
    // received a beacon during channel scanning
    if (call BeaconFrame.parsePANDescriptor(
          frame, RADIO_CHANNEL, page, &m_PANDescriptor) == SUCCESS) {
      // let's see if the beacon is from our coordinator...
      if (m_PANDescriptor.CoordAddrMode == ADDR_MODE_SHORT_ADDRESS &&
          m_PANDescriptor.CoordPANId == PAN_ID &&
          m_PANDescriptor.CoordAddress.shortAddress == COORDINATOR_ADDRESS){
        // yes! wait until SCAN is finished, then syncronize to the beacons
        m_wasScanSuccessful = TRUE;
      }
    }
  } else {
    // received a beacon during synchronization, toggle LED2
    if (beaconSequenceNumber & 1)
      call Leds.led2On();
    else
      call Leds.led2Off();
  }

  return frame;
}

event void MLME_SCAN.confirm    (
                        ieee154_status_t status,
                        uint8_t ScanType,
                        uint8_t ChannelPage,
                        uint32_t UnscannedChannels,
                        uint8_t EnergyDetectListNumEntries,
                        int8_t* EnergyDetectList,
                        uint8_t PANDescriptorListNumEntries,
                        ieee154_PANDescriptor_t* PANDescriptorList
                      )
{
  if (m_wasScanSuccessful) {
    // we received a beacon from the coordinator before
    call MLME_SET.macCoordShortAddress(m_PANDescriptor.CoordAddress.shortAddress);
    call MLME_SET.macPANId(m_PANDescriptor.CoordPANId);
    call MLME_SYNC.request(m_PANDescriptor.LogicalChannel, m_PANDescriptor.ChannelPage, TRUE);
    call MLME_SET.macRxOnWhenIdle(TRUE);
    call Frame.setAddressingFields(
        &m_frame,
        ADDR_MODE_SHORT_ADDRESS,        // SrcAddrMode,
        ADDR_MODE_SHORT_ADDRESS,        // DstAddrMode,
        m_PANDescriptor.CoordPANId,     // DstPANId,
        &m_PANDescriptor.CoordAddress,  // DstAddr,
        NULL                            // security
        );
```

Figure A.26: IRSensorC.nc

```
        post getData();
        call Timer1.startPeriodicAt(2400,Ts);
   } else
     startApp();
 }

 event void MCPS_DATA.confirm    (
                        message_t *msg,
                        uint8_t msduHandle,
                        ieee154_status_t status,
                        uint32_t timestamp
                      )
 {
   if (status == IEEE154_SUCCESS && m_ledCount++ >= 2) {
     m_ledCount = 0;
     call Leds.led1Toggle();
   }
  // previous
  //post packetSendTask();
 }

 event void MLME_SYNC_LOSS.indication(
                        ieee154_status_t lossReason,
                        uint16_t PANId,
                        uint8_t LogicalChannel,
                        uint8_t ChannelPage,
                        ieee154_security_t *security)
 {
   m_wasScanSuccessful = FALSE;
   call Leds.led1Off();
   call Leds.led2Off();
   call Timer0.stop();
   call Timer1.stop();

   startApp();
 }

 event message_t* MCPS_DATA.indication (message_t* frame)
  {
    if (call Frame.getPayloadLength(frame) == m_payloadLen && call Frame.getFrameType(frame) == FRAMETYPE_DATA) {
        MyMsg *receivepkt = (MyMsg*) (call Packet.getPayload(frame,m_payloadLen ));

    if(receivepkt->srcId==BASE_NODEID){

       if(receivepkt->other==3 && receivepkt->trgtId==111){
               call Leds.led0Toggle();
               call Timer1.stop();
            }
            else if(receivepkt->other==4 && receivepkt->trgtId==TOS_NODE_ID){
               Ts=receivepkt->data[0];
            }
      }
    }
    return frame;
  }
}
```

Figure A.27: IRSensorC.nc

IntermediateNodeAppC.nc: code for the coordinator mote

```
#include <Timer.h>
#include "printf.h"
#include "IntermediateNode.h"
#include "TKN154.h"
#include "app_profile.h"

configuration IntermediateNodeAppC {
}
implementation
{
  components MainC,
    IntermediateNodeC,
    LedsC,  Ieee802154BeaconEnabledC as MAC;

  //connections
  IntermediateNodeC -> MainC.Boot;
  IntermediateNodeC.Leds -> LedsC;
  IntermediateNodeC.MLME_START -> MAC;
  IntermediateNodeC.MCPS_DATA -> MAC;
  IntermediateNodeC.Frame -> MAC;

  IntermediateNodeC.MLME_RESET -> MAC;
  IntermediateNodeC.MLME_SET -> MAC;
  IntermediateNodeC.MLME_GET -> MAC;
  IntermediateNodeC.IEEE154TxBeaconPayload -> MAC;
  IntermediateNodeC.Packet -> MAC;

}
```

Figure A.28: IntermediateNodeAppC.nc

IntermediateNodeC.nc: code for the coordinator mote

```
#include "AM.h"
#include <Timer.h>
#include "printf.h"
#include "IntermediateNode.h"

#include "TKN154.h"
#include "app_profile.h"

module IntermediateNodeC
{
uses {
interface Boot;
interface Leds;
interface Packet;
interface MCPS_DATA;
interface MLME_RESET;
interface MLME_START;
interface MLME_SET;
interface MLME_GET;
interface IEEE154Frame as Frame;
interface IEEE154TxBeaconPayload;
}
}

implementation
{

bool m_ledCount;
MyMsg* receivepkt;
message_t m_frame;
uint8_t m_payloadLen = sizeof(MyMsg);

void setAddressingFields(uint16_t address);

event void Boot.booted() {
call MLME_RESET.request(TRUE);
}

/*****************************************************************
 * IEEE 802.15.4
 *****************************************************************/
void setAddressingFields(uint16_t address)
{
ieee154_address_t deviceShortAddress;
deviceShortAddress.shortAddress = address; // destination

call Frame.setAddressingFields(
&m_frame,
ADDR_MODE_SHORT_ADDRESS, // SrcAddrMode,
ADDR_MODE_SHORT_ADDRESS, // DstAddrMode,
PAN_ID, // DstPANId,
&deviceShortAddress, // DstAddr,
NULL // security
);
}

event void MLME_RESET.confirm(ieee154_status_t status)
{
if (status != IEEE154_SUCCESS)
return;
call MLME_SET.phyTransmitPower(TX_POWER);
call MLME_SET.macShortAddress(INTERMEDIATE_NODEID);
```

Figure A.29: IntermediateNodeC.nc

```
call MLME_SET.macAssociationPermit(FALSE);
call MLME_START.request(
PAN_ID, // PANId
RADIO_CHANNEL, // LogicalChannel
0, // ChannelPage,
0, // StartTime,
BEACON_ORDER, // BeaconOrder
SUPERFRAME_ORDER, // SuperframeOrder
TRUE, // PANCoordinator
FALSE, // BatteryLifeExtension
FALSE, // CoordRealignment
0, // CoordRealignSecurity,
0 // BeaconSecurity
);
}

event message_t* MCPS_DATA.indication ( message_t* frame )
{
if (m_ledCount++ == 20) {
m_ledCount = 0;
call Leds.led1Toggle();
}
if (call Frame.getPayloadLength(frame) == m_payloadLen &&
call Frame.getFrameType(frame) == FRAMETYPE_DATA) {
uint16_t target;
uint8_t i;
uint8_t* payloadRegion;

receivepkt = (MyMsg*) (call Packet.getPayload(frame,m_payloadLen));
target=receivepkt->trgtId;

if(target==INTERMEDIATE_NODEID) {
receivepkt->trgtId=BASE_NODEID;
setAddressingFields(BASE_NODEID);

} else if(target==BASE_NODEID) {
return frame;
} else {/*(target!=INTERMEDIATE_NODEID)*/
if(receivepkt->other==3 && receivepkt->trgtId==111) {
i=0x0015;
setAddressingFields(i);
payloadRegion = call Packet.getPayload(&m_frame, m_payloadLen);
if (m_payloadLen <= call Packet.maxPayloadLength()) {
memcpy(payloadRegion, receivepkt, m_payloadLen);
}

for (i=21 ; i < 33 ; i++ ){
if (i==29 || i==30 || i ==31){}
else{
setAddressingFields(i);
if (call MCPS_DATA.request (
&m_frame, // frame,
m_payloadLen, // payloadLength,
0, // msduHandle,
TX_OPTIONS_ACK // TxOptions,
) != IEEE154_SUCCESS)
call Leds.led0Toggle(); //fail!
else
call Leds.led0Off();
}
}
return frame;
}
if( 20 < receivepkt->trgtId && receivepkt->trgtId < 33) {
setAddressingFields(target);
}
}
```

Figure A.30: IntermediateNodeC.nc

```
payloadRegion = call Packet.getPayload(&m_frame, m_payloadLen);
if (m_payloadLen <= call Packet.maxPayloadLength()) {
memcpy(payloadRegion, receivepkt, m_payloadLen);
}

if (call MCPS_DATA.request (
&m_frame, // frame,
m_payloadLen, // payloadLength,
0, // msduHandle,
TX_OPTIONS_ACK // TxOptions,
) != IEEE154_SUCCESS)
call Leds.led0Toggle(); //fail!
else
call Leds.led0Off();
}
if (m_ledCount++ == 20) {
m_ledCount = 0;
call Leds.led1Toggle();
}
return frame;
}

event void MLME_START.confirm(ieee154_status_t status) {}

event void MCPS_DATA.confirm(
message_t *msg,
uint8_t msduHandle,
ieee154_status_t status,
uint32_t Timestamp
) {
}

event void IEEE154TxBeaconPayload.aboutToTransmit() {}

event void IEEE154TxBeaconPayload.setBeaconPayloadDone(void *beaconPayload, uint8_t length) {}

event void IEEE154TxBeaconPayload.modifyBeaconPayloadDone(uint8_t offset, void *buffer, uint8_t bufferLength) {}

event void IEEE154TxBeaconPayload.beaconTransmitted()
{
ieee154_macBSN_t beaconSequenceNumber = call MLME_GET.macBSN();
if (beaconSequenceNumber & 1)
call Leds.led2On();
else
call Leds.led2Off();
}
}
```

Figure A.31: IntermediateNodeC.nc

BaseStationC.nc: code for the base mote

```
/*
 * BaseStation154C.nc
 *
 * KTH | Royal Institute of Technology
 * Automatic Control
 *
 *           Project: se.kth.tinyos2x.ieee802154.tkn154
 *        Created on: 2010/05/10
 * Last modification: 2010/04/19
 *            Author: aitorhh
 *
 */
#include "app_profile.h"

configuration BaseStationC {
}implementation {
  components MainC, LedsC;
components Ieee802154BeaconEnabledC as MAC;

  components BaseStationP as App,
             SerialActiveMessageC as Serial;

  App.MLME_START -> MAC;
  App.MCPS_DATA -> MAC;
  App.Frame -> MAC;
  App.BeaconFrame -> MAC;
  App.Packet -> MAC;

  MainC.Boot <- App;
  App.Leds -> LedsC;
  App.MLME_RESET -> MAC;
  App.MLME_SET -> MAC;
  App.MLME_GET -> MAC;

  App.SerialControl -> Serial;
  App.UartSend -> Serial.AMSend[AM_MYMSG];
  App.UartReceive -> Serial.Receive[AM_MYMSG];
  App.UartAMPacket -> Serial;

  App.MLME_SCAN -> MAC;
  App.MLME_SYNC -> MAC;
  App.MLME_BEACON_NOTIFY -> MAC;
  App.MLME_SYNC_LOSS -> MAC;
}
```

Figure A.32: BaseStationC.nc

BaseStationP.nc: code for the base mote

```
/*
 * BaseStation154C.nc
 *
 * KTH | Royal Institute of Technology
 * Automatic Control
 *
 *           Project: se.kth.tinyos2x.ieee802154.tkn154
 *        Created on: 2010/05/10
 * Last modification:
 *            Author: aitorhh
 *
 */

#include "AM.h"
#include "Serial.h"
#include "TKN154.h"

#include "app_profile.h"

module BaseStationP {
uses {
interface Boot;
interface MCPS_DATA;
interface MLME_RESET;
interface MLME_START;
interface MLME_SET;
interface MLME_GET;
interface IEEE154Frame as Frame;
interface Leds;
interface Packet;

interface MLME_SCAN;
interface MLME_SYNC;
interface MLME_BEACON_NOTIFY;
interface MLME_SYNC_LOSS;
interface IEEE154BeaconFrame as BeaconFrame;

interface SplitControl as SerialControl;
interface AMSend as UartSend;
interface Receive as UartReceive;
interface AMPacket as UartAMPacket;
}
}
implementation {

message_t m_frame;
am_id_t id = AM_MYMSG;
MyMsg* datapkt;

uint8_t counter;
uint8_t m_payloadLen = sizeof(MyMsg);

// to be a device
ieee154_PANDescriptor_t m_PANDescriptor;
bool m_ledCount;
bool m_wasScanSuccessful;

void setAddressingFields(message_t *msg,uint16_t address);

event void Boot.booted() {

call SerialControl.start();
```

Figure A.33: BaseStationP.nc

```
datapkt =(MyMsg*)(call Packet.getPayload(&m_frame,sizeof(MyMsg)));
datapkt->srcId = TOS_NODE_ID;
datapkt->trgtId = 111;
datapkt->other = 3;
call MLME_RESET.request(TRUE);
}

void setAddressingFields(message_t *msg,uint16_t address)
{
ieee154_address_t deviceShortAddress;
deviceShortAddress.shortAddress = address; // destination
call Frame.setAddressingFields(
msg,
ADDR_MODE_SHORT_ADDRESS, // SrcAddrMode,
ADDR_MODE_SHORT_ADDRESS, // DstAddrMode,
PAN_ID, // DstPANId,
&deviceShortAddress, // DstAddr,
NULL // security
);
}

void startApp()
{
ieee154_phyChannelsSupported_t channelMask;
uint8_t scanDuration = BEACON_ORDER;

call MLME_SET.phyTransmitPower(TX_POWER);
call MLME_SET.macShortAddress(TOS_NODE_ID);

// scan only the channel where we expect the coordinator
channelMask = ((uint32_t) 1) << RADIO_CHANNEL;

// we want all received beacons to be signalled
// through the MLME_BEACON_NOTIFY interface, i.e.
// we set the macAutoRequest attribute to FALSE
call MLME_SET.macAutoRequest(FALSE);
m_wasScanSuccessful = FALSE;
call MLME_SCAN.request (
PASSIVE_SCAN, // ScanType
channelMask, // ScanChannels
scanDuration, // ScanDuration
0x00, // ChannelPage
0, // EnergyDetectListNumEntries
NULL, // EnergyDetectList
0, // PANDescriptorListNumEntries
NULL, // PANDescriptorList
0 // security
);
}

event message_t* MLME_BEACON_NOTIFY.indication (message_t* frame)
{
// received a beacon frame
ieee154_phyCurrentPage_t page = call MLME_GET.phyCurrentPage();

if (!m_wasScanSuccessful) {
// received a beacon during channel scanning
if (call BeaconFrame.parsePANDescriptor(
frame, RADIO_CHANNEL, page, &m_PANDescriptor) == SUCCESS) {
// let's see if the beacon is from our coordinator...
if (m_PANDescriptor.CoordAddrMode == ADDR_MODE_SHORT_ADDRESS &&
m_PANDescriptor.CoordPANId == PAN_ID &&
m_PANDescriptor.CoordAddress.shortAddress == COORDINATOR_ADDRESS) {
// yes! wait until SCAN is finished, then syncronize to the beacons
m_wasScanSuccessful = TRUE;
}
}
```

Figure A.34: BaseStationP.nc

```
} else {
// received a beacon during synchronization, toggle LED2
ieee154_macBSN_t beaconSequenceNumber = call BeaconFrame.getBSN(frame);

if (beaconSequenceNumber & 1)
call Leds.led2On();
else
call Leds.led2Off();
}

return frame;
}

event void MLME_SCAN.confirm (
ieee154_status_t status,
uint8_t ScanType,
uint8_t ChannelPage,
uint32_t UnscannedChannels,
uint8_t EnergyDetectListNumEntries,
int8_t* EnergyDetectList,
uint8_t PANDescriptorListNumEntries,
ieee154_PANDescriptor_t* PANDescriptorList
)
{
if (m_wasScanSuccessful) {
// we received a beacon from the coordinator before
call MLME_SET.macCoordShortAddress(m_PANDescriptor.CoordAddress.shortAddress);
call MLME_SET.macPANId(m_PANDescriptor.CoordPANId);
call MLME_SYNC.request(m_PANDescriptor.LogicalChannel, m_PANDescriptor.ChannelPage, TRUE);

call Frame.setAddressingFields(
&m_frame,
ADDR_MODE_SHORT_ADDRESS, // SrcAddrMode,
ADDR_MODE_SHORT_ADDRESS, // DstAddrMode,
m_PANDescriptor.CoordPANId, // DstPANId,
&m_PANDescriptor.CoordAddress, // DstAddr,
NULL // security
);
                if (call MCPS_DATA.request ( &m_frame, // frame,
sizeof(MyMsg), // payloadLength,
0, // msduHandle,
TX_OPTIONS_ACK // TxOptions,
) != IEEE154_SUCCESS) {
call Leds.led0Toggle(); //fail!
} else {
call Leds.led0Off();
}
} else
startApp();
}

event void MLME_SYNC_LOSS.indication(
ieee154_status_t lossReason,
uint16_t PANId,
uint8_t LogicalChannel,
uint8_t ChannelPage,
ieee154_security_t *security)
{
m_wasScanSuccessful = FALSE;
call Leds.led1Off();
call Leds.led2Off();
startApp();
}

event void MLME_RESET.confirm(ieee154_status_t status)
{
if (status != IEEE154_SUCCESS) return;
```

Figure A.35: BaseStationP.nc

```
call MLME_SET.phyTransmitPower(TX_POWER);
call MLME_SET.macShortAddress(TOS_NODE_ID);
call MLME_SET.macAssociationPermit(FALSE);
call MLME_SET.macRxOnWhenIdle(TRUE);

call MLME_START.request(
PAN_ID, // PANId
RADIO_CHANNEL, // LogicalChannel
0, // ChannelPage,
0, // StartTime,
BEACON_ORDER, // BeaconOrder
SUPERFRAME_ORDER, // SuperframeOrder
TRUE, // PANCoordinator
FALSE, // BatteryLifeExtension
FALSE, // CoordRealignment
0, // CoordRealignSecurity,
0 // BeaconSecurity
);
startApp();
}
event void MLME_START.confirm(ieee154_status_t status) {}

/*****************************************************************************
 *  DEVICE -----15.4----- BASE STATION -------MyMsg------- PC
 *****************************************************************************/

//Store the new message in the UART input buffer
event message_t* MCPS_DATA.indication (message_t* frame)
{
// We assume that the packets with length equal to MyMsg
// are a MyMsg
if (call Frame.getPayloadLength(frame) == m_payloadLen &&
call Frame.getFrameType(frame) == FRAMETYPE_DATA) {

if (call UartSend.send(AM_BROADCAST_ADDR, frame, sizeof(MyMsg) ) == SUCCESS) {
call Leds.led1Toggle();
} else {
call Leds.led0Toggle();
}
}
return frame;
}

event void SerialControl.startDone(error_t error) {}

event void SerialControl.stopDone(error_t error) {}

event void UartSend.sendDone(message_t* msg, error_t error) {}

/*****************************************************************************
 *  PC -----MyMsg----- BASE SATITON -------15.4------- DEVICE
 *****************************************************************************/
event message_t *UartReceive.receive(message_t *msg,
void *payload,
uint8_t len) {
return msg;
}

event void MCPS_DATA.confirm (
message_t *msg,
uint8_t msduHandle,
ieee154_status_t status,
uint32_t timestamp
) {}
task void sendRatePacket() {}
}
```

Figure A.36: BaseStationP.nc

Appendix B

Matlab code for the treatment of the data and the pattern detection

Matlab code for the treatment of the data and the pattern detection

It is divided in four parts depending on which analysis to perform.

Plots.m: code to plot the data of the sensors, it is possible to plot as many sensors as wanted alltogether.

```
close all

%D is a matrix where the first column is the data, the second one is a
%parameter that tells us which kind of sensor it is or if it corresponds to
%the plug1 or plug2. the third column is the number of the mote. the fourth
%is the hour, the fifth the minute and the sixth the second. Then the
%columns from 7 to 15 are the 9 other values of data that only the
%vibration sensor has, these columns are 0 for the rest of sensors.
D=sortrows(simout.signals.values,[2 3]);
[lengthD,widthD]=size(D);

%The values equal to D(i,2) and D(i,3) must be changed depending on what to
%plot.
%Powerplug1 DW: D(i,2)==1 && D(i,3)==21
%Powerplug1 MW: D(i,2)==2 && D(i,3)==21
%Powerplug2 CF: D(i,2)==1 && D(i,3)==22
%Powerplug2 WB: D(i,2)==2 && D(i,3)==22
%People door1: D(i,2)==7 && D(i,3)==24
%People door2: D(i,2)==7 && D(i,3)==25
%Temp mote 1: D(i,2)==6 && D(i,3)==26
%Temp mote 2: D(i,2)==6 && D(i,3)==27
%Light mote 1: D(i,2)==5 && D(i,3)==26
%Light mote 2: D(i,2)==5 && D(i,3)==27
%Fridge door: D(i,2)==0 && D(i,3)==23
%Water consumption: D(i,2)==0 && D(i,3)==28

%For interconnected powerplugs:
%Powerplug1 CF: D(i,2)==1 && D(i,3)==22
%Powerplug1 WB: D(i,2)==2 && D(i,3)==22
%Powerplug2 MW: D(i,2)==2 && D(i,3)==21
%Powerplug2 CM+WB: D(i,2)==1 && D(i,3)==21
%Powerplug3 DW: D(i,2)==2 && D(i,3)==31
%Powerplug3 CM+WB+MW: D(i,2)==1 && D(i,3)==31
%Powerplug4 ALL: D(i,2)==1 && D(i,3)==33

%v2=[1 2 1 2 1 2 1 2 7 7 6 6 5 5 0 0 7]
%v3=[21 21 22 22 31 31 33 33 24 25 26 27 26 27 23 28 32]

v2=[1 2 1 2 7 7 7 0 0]
v3=[22 22 21 21 24 25 32 23 28]


[rows,lengthv]=size(v2)
lengthPLOT=lengthv;
%PLOTtitles=['
figure,
mm=1;
lost=0;
for m=1:lengthPLOT
    m
i=1;
k=1;
n=1;
clear a
    while i<=lengthD
        if D(i,2)==v2(1,m) && D(i,3)==v3(1,m)
            for j=1:15
                a(k,j)=D(i,j);
            end
            a(k,16)=0;%column that tells us if it's a lost package that we added or not (1 when lost)
```

Figure B.1: Plots.m

```
        a(k,4)=a(k,4)*100 +a(k,5)/0.6+a(k,6)/0.6/100/0.6;
        if  k>1
            if a(k,2)==6 %if the lost packet is for the temp sensor, their sampling rate is 60s
              if(a(k,4)-a(k-1,4))>1.8
 lost=lost+1; %counts the number of lost packages
                 for j=1:15
                     a(k+1,j)=a(k,j);
                     a(k,j)=a(k-1,j);
                     a(k,16)=1;
                 end
                 a(k,4)=a(k,4)+1.7;
                 a(k,5)=a(k,5)+1;
                 if a(k,5)>=60
                     a(k,5)=a(k,5)-60;
                     a(k,6)=a(k,6)+1;
                 end
                 i=i-1;
                 %k=k+1;
              end
            else
              if(a(k,4)-a(k-1,4))>0.3 %for the rest of sensors sampling rate 10s
                 lost=lost+1; %counts the number of lost packages
                 for j=1:15
                     %a(k+1,j)=a(k,j);
                     a(k,j)=a(k-1,j);
                     a(k,16)=1;
                 end
                 a(k,4)=a(k,4)+0.28;
                 a(k,6)=a(k,6)+10;
                 if a(k,6)>=60
                     a(k,6)=a(k,6)-60;
                     a(k,5)=a(k,5)+1;
                 end
                 if a(k,5)>=60
                     a(k,5)=a(k,5)-60;
                     a(k,6)=a(k,6)+1;
                 end
                 i=i-1;
                 %k=k+1;
              end
            end
        end
        k=k+1;
    end
    i=i+1;
end

[lengthA,widthA]=size(a);

%if we have a vibration or a current sensor we will create another matrix b and we
%will plot that one
if ((a(1,2)==0 || a(1,2)==1 ||a(1,2)==2) && (a(1,3)==23 || a(1,3)==28 || a(1,3)==21 || a(1,3)==31 || a(1,3)==33)) || (a(1,2)==2 && a(1,3)==22) %if the data is
    clear b
    for i=1:lengthA %copy the rest of the data to the matrix b
        b(n,1)=a(i,1);
        b(n,2)=a(i,4)-0.1/0.6; %correcting the clock
        for j=7:15
               b(n+j-6,1)=a(i,j);
               b(n+j-6,2)=a(i,4)-0.1/0.6+(j-6)*0.01/0.6/0.6; %correcting the clock
        end
        n=n+10;
    end
    [lengthB,widthB]=size(b);
    for i=1:lengthB
        if b(i,1)>4096
            b(i,1)=4096;
        end
```

Figure B.2: Plots.m

```
        if b(i,1)<-4096
            b(i,1)=-4096;
        end
    end
end
subplot(lengthPLOT,1,m)

%treat the data, we'll plot the events showing a 1 when something goes
%on and a 0 when doesn't
%for the coffee machine:

k=0;
j=1;

if (a(1,2)==1) && (a(1,3)==22)
    clear coffee
    for i=1:lengthA
        if a(i,1)>200    %200 is just a threshold value
            coffee(j,2)=1;
            coffee(j,1)=a(i,4);
            j=j+1;
        else
            coffee(j,2)=0;
            coffee(j,1)=a(i,4);
            j=j+1;
        end
    end
    [lengthCoffee,widthCoffee]=size(coffee);
end

%for the water boiler:
k=0;
j=1;

if (a(1,2)==2) && (a(1,3)==22)
    clear boiler
    for i=1:lengthB
        if b(i,1)>4    %4 is just a threshold value
            boiler(j,2)=1;
            boiler(j,1)=b(i,2);
            j=j+1;
        else
            boiler(j,2)=0;
            boiler(j,1)=b(i,2);
            j=j+1;
        end
    end
    [lengthBoiler,widthBoiler]=size(boiler);
end

 %for the dishwasher:
k=0;
j=1;

if (a(1,2)==1) && (a(1,3)==21)
    clear dishwasher
    for i=1:lengthB
        if b(i,1)>4    %4 is just a threshold value
            dishwasher(j,2)=1;
            dishwasher(j,1)=b(i,2);
            j=j+1;
        else
            dishwasher(j,2)=0;
            dishwasher(j,1)=b(i,2);
            j=j+1;
        end
    end
```

Figure B.3: Plots.m

```
    [lengthDishwasher,widthDishwasher]=size(dishwasher);
end

%for the microwave:
k=0;
j=1;

if (a(1,2)==2) && (a(1,3)==21)
    for i=1:lengthB
        if b(i,1)>4    %4 is just a threshold value
            microwave(j,2)=1;
            microwave(j,1)=b(i,2);
            j=j+1;
        else
            microwave(j,2)=0;
            microwave(j,1)=b(i,2);
            j=j+1;
        end
    end
    [lengthMicrowave,widthMicrowave]=size(microwave);
end

%for the sink vibrational sensor
%we will plot a 1 when someone is using water and a -1 when someone
%puts a cup in the sink

k=0;
j=1;
l=0;

if (a(1,2)==0) && (a(1,3)==28)
    clear sink
    for i=1:lengthB
        if b(i,1)>3600 || b(i,1)<2800  %these numbers are thresholds
            sink(j,2)=-1;
            sink(j,1)=b(i,2);
            j=j+1;
        elseif 3400>b(i,1) && b(i,1)>3300
            sink(j,2)=0;
            sink(j,1)=b(i,2);
            j=j+1;
        elseif (3600>b(i,1) && b(i,1)>3400) || (2800<b(i,1) && b(i,1)<3300)
            sink(j,2)=1;
            sink(j,1)=b(i,2);
            j=j+1;
        end
    end
    [lengthSink,widthSink]=size(sink);
end

%for the fridge vibrational sensor
%we will plot a 1 when someone opens or -1 when someone closes the door

k=0;
j=1;
l=0;

if (a(1,2)==0) && (a(1,3)==23)
    clear fridge
    for i=1:lengthB
        if b(i,1)>3450     %these numbers are thresholds open the fridge
            fridge(j,2)=-1;
            fridge(j,1)=b(i,2);
            j=j+1;
        elseif b(i,1)>3400 && b(i,1)<=3450   %these numbers are thresholds close the fridge
            fridge(j,2)=1;
            fridge(j,1)=b(i,2);
```

Figure B.4: Plots.m

```
            j=j+1;
        else
            fridge(j,2)=0;
            fridge(j,1)=b(i,2);
            j=j+1;
        end
    end
    [lengthFridge,widthFridge]=size(fridge);
end

%for the IRSensor in the doors, when plot positives numbers when people
%enters and negative when they leave

k=0;
j=1;
l=0;

if (a(1,2)==7) && (a(1,3)==24)
    clear IR1
    for i=1:lengthA
        IR1(j,2)=a(i,1)-100;
        IR1(j,1)=a(i,4);
        j=j+1;
    end
    [lengthIR1,widthIR1]=size(IR1);
end

if (a(1,2)==7) && (a(1,3)==25)
    clear IR2
    for i=1:lengthA
        IR2(j,2)=a(i,1)-100;
        IR2(j,1)=a(i,4);
        j=j+1;
    end
    [lengthIR2,widthIR2]=size(IR2);
end

if (a(1,2)==7) && (a(1,3)==32)
    clear IR3
    for i=1:lengthA
        IR3(j,2)=a(i,1)-100;
        IR3(j,1)=a(i,4);
        j=j+1;
    end
    [lengthIR3,widthIR3]=size(IR3);
end

if (a(1,2)==7) && (a(1,3)==32)
    clear IR
    lengthIR=lengthIR1;
    if lengthIR2<lengthIR
        lengthIR=lengthIR2;
    end
    if lengthIR3<lengthIR
        lengthIR=lengthIR3;
    end
    j=1;
    for i=1:lengthIR
        IR(j,2)=IR1(i,2)+IR2(i,2)+IR3(i,2);
        %if IR(j,2)<0
        %    IR(j,2)=0;
        %end
        IR(j,1)=IR1(i,1);
        j=j+1;
    end
    [lengthIR,widthIR]=size(IR);
end
```

Figure B.5: Plots.m

```
    if (a(1,2)==2 ||a(1,2)==2) && (a(1,3)==31)
        plot(b(1:lengthB,2),b(1:lengthB,1), 'DisplayName', 'b(1:lengthB,1)', 'YDataSource', 'b(1:lengthB,1)');
    elseif (a(1,2)==0 && a(1,3)==23)
        plot(fridge(1:lengthFridge,1),fridge(1:lengthFridge,2), 'DisplayName', 'fridge(1:lengthFridge,2)', 'YDataSource', 'fridge(1:lengthFridge,2)');
    elseif (a(1,2)==0 && a(1,3)==28)
        plot(sink(1:lengthSink,1),sink(1:lengthSink,2), 'DisplayName', 'sink(1:lengthSink,1)', 'YDataSource', 'sink(1:lengthSink,2)');
    elseif (a(1,2)==1 && a(1,3)==22)
        plot(coffee(1:lengthCoffee,1),coffee(1:lengthCoffee,2), 'DisplayName', 'coffee(1:lengthCoffee,2)', 'YDataSource', 'coffee(1:lengthCoffee,2)');
    elseif (a(1,2)==2 && a(1,3)==22)
        plot(boiler(1:lengthBoiler,1),boiler(1:lengthBoiler,2), 'DisplayName', 'boiler(1:lengthBoiler,2)', 'YDataSource', 'boiler(1:lengthBoiler,2)');
    %elseif (a(1,2)==7 && a(1,3)==24)
    %    plot(IR1(1:lengthIR1,1),IR1(1:lengthIR1,2), 'DisplayName', 'IR1(1:lengthIR1,2)', 'YDataSource', 'IR1(1:lengthIR1,2)');
    %elseif (a(1,2)==7 && a(1,3)==25)
    %    plot(IR2(1:lengthIR2,1),IR2(1:lengthIR2,2), 'DisplayName', 'IR2(1:lengthIR2,2)', 'YDataSource', 'IR2(1:lengthIR2,2)');
    elseif (a(1,2)==7 && a(1,3)==32)
        %plot(IR3(1:lengthIR3,2),IR3(1:lengthIR3,1), 'DisplayName', 'IR3(1:lengthIR3,2)', 'YDataSource', 'IR3(1:lengthIR3,2)');
        plot(IR(1:lengthIR,1),IR(1:lengthIR,2), 'DisplayName', 'IR(1:lengthIR,2)', 'YDataSource', 'IR(1:lengthIR,2)');
    elseif (a(1,2)==2 && a(1,3)==21)
        plot(microwave(1:lengthMicrowave,1),microwave(1:lengthMicrowave,2), 'DisplayName', 'microwave(1:lengthMicrowave,2)', 'YDataSource', 'microwave(1:lengthMicrowave,2)');
    elseif (a(1,2)==1 && a(1,3)==21)
        plot(dishwasher(1:lengthDishwasher,1),dishwasher(1:lengthDishwasher,2), 'DisplayName', 'dishwasher(1:lengthDishwasher,2)', 'YDataSource', 'dishwasher(1:lengthDishwasher,2)');
    elseif (a(1,2)==1 && a(1,3)==31)
        plot(CF_WB(1:lengthCF_WB,1),CF_WB(1:lengthCF_WB,2), 'DisplayName', 'CF_WB(1:lengthCF_WB,2)', 'YDataSource', 'CF_WB(1:lengthCF_WB,2)');
    end


    %title(PLOTtitles)
    %here we choose which matrix we plot depending on the kind of sensor
    %if (a(1,2)==1 || a(1,2)==6) && (a(1,3)==22 || a(1,3)==27)
    %    plot(b(1:lengthB,2),b(1:lengthB,1), 'DisplayName', 'b(1:lengthb,1)', 'YDataSource', 'b(1:lengthB,1)');
    %else
    %    plot(a(1:lengthA,4),a(1:lengthA,1), 'DisplayName', 'a(1:lengthA,1)', 'YDataSource', 'a(1:lengthA,1)');
    %end
    if mm==1
    v=axis
    mm=2;
    end
    %axis(v(1));
    %xlim([0 1200])
    hold on
end
```

Figure B.6: Plots.m

Eventsvector.m: creates the binary vectors for each one of the sensors

```
%%% CREATE THE EVENTS VECTOR FROM THE MATRIX %%%

% we will create three vectors that contain the sequence of all the events
% that happened. then just checking the order of these events we can guess
% what was going on in the kitchen

j=1;
clear E %vector of events
clear T %vector with the time of each corresponding event
clear P %vector with the number of people in the room at each event

for i=2:widthM

    if M(2,i)>M(2,i-1)
        E(j)='b';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    elseif M(2,i)<M(2,i-1)
        E(j)='v';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    end

    if M(3,i)>M(3,i-1)
        E(j)='c';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    elseif M(3,i)<M(3,i-1)
        E(j)='e';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    end

    if M(4,i)>M(4,i-1)
        E(j)='m';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    elseif M(4,i)<M(4,i-1)
        E(j)='i';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    end

    if M(5,i)>M(5,i-1)
        E(j)='d';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    elseif M(5,i)<M(5,i-1)
        E(j)='h';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    end
```

Figure B.7: Eventsvector.m

```
    if M(6,i)>M(6,i-1)
        if M(6,i)~=0
            E(j)='g';
            T(j)=M(1,i);
            P(j)=M(8,i);
            j=j+1;
        end
    elseif M(6,i)<M(6,i-1)
        if M(6,i)~=0
            E(j)='f';
            T(j)=M(1,i);
            P(j)=M(8,i);
            j=j+1;
        end
    end

    if M(7,i)>M(7,i-1)
        if M(7,i)~=0
            E(j)='w';
            T(j)=M(1,i);
            P(j)=M(8,i);
            j=j+1;
        end
    elseif M(7,i)<M(7,i-1)
        if M(7,i)~=0
            E(j)='s';
            T(j)=M(1,i);
            P(j)=M(8,i);
            j=j+1;
        end
    end

    if M(8,i)>M(8,i-1)
        E(j)='E';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    elseif M(8,i)<M(8,i-1)
        E(j)='L';
        T(j)=M(1,i);
        P(j)=M(8,i);
        j=j+1;
    end
end

[lengthE,widthE]=size(E);

%%% WE SHOW ALL THE EVENTS THAT HAPPENED AND COUNT SOME STATISTICS %%%

nb=0;
nc=0;
nm=0;
nd=0;
nw=0;
ns=0;
ne=0;
nl=0;
nf=0;
```

Figure B.8: Eventsvector.m

```
for j=1:widthE
   if E(j)=='b'
       disp(sprintf('At %d:%d boiler started',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
       nb=nb+1;
   elseif E(j)=='v'
       disp(sprintf('At %d:%d boiler finished',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
   elseif E(j)=='c'
       disp(sprintf('At %d:%d coffee machine started',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
       nc=nc+1;
   elseif E(j)=='e'
       disp(sprintf('At %d:%d coffee machine finished',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
   elseif E(j)=='m'
       disp(sprintf('At %d:%d microwave started',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
       nm=nm+1;
   elseif E(j)=='i'
       disp(sprintf('At %d:%d microwave finished',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
   elseif E(j)=='d'
       disp(sprintf('At %d:%d dishwasher started',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
       nd=nd+1;
   elseif E(j)=='h'
       disp(sprintf('At %d:%d dishwasher finished',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
       nd=nd+1;
   elseif E(j)=='w'
       disp(sprintf('At %d:%d water',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
       nw=nw+1;
   elseif E(j)=='s'
       disp(sprintf('At %d:%d sink',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
       ns=ns+1;
   elseif E(j)=='f'
       disp(sprintf('At %d:%d fridge opened',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
       nf=nf+1;
   elseif E(j)=='g'
       disp(sprintf('At %d:%d fridge closed',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
   elseif E(j)=='E'
       disp(sprintf('At %d:%d entered',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
       ne=ne+1;
   elseif E(j)=='L'
       disp(sprintf('At %d:%d left',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
       nl=nl+1;
   end
end
```

Figure B.9: Eventsvector.m

Matrixofevents.m: creates the binary matrix that includes all the events

```
%%% CREATE THE MATRIX WITH ALL THE DATA IN IT %%%%
% we will save the time in the first row and then the data from the
% diferent sensors in each following row. To do this we'll be checking the
% time of each data and saving it in the matrix in a position that the time
% fits with an error of less than 1 second. We have sensors that their data
% is every second and other that are every 10 seconds, so we will fill all
% the time positions in the middle with the last real value of the sensor.

if(lengthPLOT==9)
    clear M

    M=boiler';
    i=1;

    for j=1:lengthCoffee
        if coffee(j,1)<2399.5
            while abs(coffee(j,1)-M(1,i))>2399 && i<lengthBoiler
                i=i+1;
            end
            while (coffee(j,1)-M(1,i))>0.015 && i<lengthBoiler
                if(j>1)
                    M(3,i)=coffee(j-1,2);
                end
                i=i+1;
            end
            if (abs(coffee(j,1)-M(1,i)))<=0.015 && i<lengthBoiler
                M(3,i)=coffee(j,2);
                i=i+1;
            end
        end
    end

    i=1;

    for j=1:lengthMicrowave
       if microwave(j,1)<2399.5
            while abs(microwave(j,1)-M(1,i))>2399 && i<lengthBoiler
                i=i+1;
            end
         while (microwave(j,1)-M(1,i))>0.015 && i<lengthBoiler
            if(j>1)
                M(4,i)=microwave(j-1,2);
            end
            i=i+1;
         end
         if (abs(microwave(j,1)-M(1,i)))<=0.015 && i<lengthBoiler
            M(4,i)=microwave(j,2);
            i=i+1;
         end
       end
    end

    i=1;

    for j=1:lengthDishwasher
        if dishwasher(j,1)<2399.5
            while abs(dishwasher(j,1)-M(1,i))>2399 && i<lengthBoiler
                i=i+1;
            end
        while (dishwasher(j,1)-M(1,i))>0.015 && i<lengthBoiler
            if(j>1)
```

Figure B.10: Matrixofevents.m

```
            M(5,i)=dishwasher(j-1,2);
        end
        i=i+1;
    end
    if (abs(dishwasher(j,1)-M(1,i)))<=0.015 && i<lengthBoiler
        M(5,i)=dishwasher(j,2);
        i=i+1;
    end
    end
end

i=1;
for j=1:lengthFridge
    if fridge(j,1)<2399.5
        while abs(fridge(j,1)-M(1,i))>2399 && i<lengthBoiler
            i=i+1;
        end
    while (fridge(j,1)-M(1,i))>0.015 && i<lengthBoiler
        if(j>1)
            M(6,i)=fridge(j-1,2);
        end
        i=i+1;
    end
    if (abs(fridge(j,1)-M(1,i)))<=0.015 && i<lengthBoiler
        M(6,i)=fridge(j,2);
        i=i+1;
    end
    end
end

i=1;
for j=1:lengthSink
   if sink(j,1)<2399.5
        while abs(sink(j,1)-M(1,i))>2399 && i<lengthBoiler
            i=i+1;
        end
    while (sink(j,1)-M(1,i))>0.015 && i<lengthBoiler
        if(j>1)
            M(7,i)=sink(j-1,2);
        end
        i=i+1;
    end
    if (abs(sink(j,1)-M(1,i)))<=0.015 && i<lengthBoiler
        M(7,i)=sink(j,2);
        i=i+1;
    end
   end
end

i=1;
for j=1:lengthIR
    if IR(j,1)<2399.5
        while abs(IR(j,1)-M(1,i))>2399 && i<lengthBoiler
            i=i+1;
        end
    while (IR(j,1)-M(1,i))>0.015 && i<lengthBoiler
       if(j>1)
            M(8,i)=IR(j-1,2);
        end
        i=i+1;
    end
    if (abs(IR(j,1)-M(1,i)))<=0.015 && i<lengthBoiler
        M(8,i)=IR(j,2);
        i=i+1;
    end
    end
end
```

Figure B.11: Matrixofevents.m

Patterndetection.m: analyses the matrix of events and identifies the patterns

```
%%% IDENTIFY THE PATTERNS FROM THE EVENTS VECTOR %%%

current_state=0;
DW_ON=0;
dishw=0;
warmwater=0;
fika=0;
DWstart=0;

for j=1:widthE

    if T(j)>1150 && T(j)<1400
        lunch_time=1;
    else
        lunch_time=0;
    end

    if E(j)=='v'
        warmwater=T(j);
    elseif E(j)=='d'
        DW_ON=1;
        if T(j)>(DWstart+50)
            DWstart=T(j);
        end
    elseif E(j)=='h'
        dishw=T(j);
        DW_ON=0;
    end



    if ((T(j)>1450 && T(j)<1550) && (P(j)>6 && P(j)<15)) && fika==0
        disp(sprintf('At %d:%d people came for fika',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        fika=1;
    elseif ((T(j)>1450 && T(j)<1550) && (P(j)>=15)) && fika==0
        disp(sprintf('At %d:%d people came for fika with cake',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        fika=1;
    end

    if T(j)>1550
        fika=0;
    end

    if current_state==0
        if E(j)=='E'
            current_state=10;
        end

    elseif current_state==10
        if E(j)=='c'
            current_state=11;
        elseif E(j)=='b'
            current_state=20;
        elseif E(j)=='d' && (T(j)-DWstart)==0
            current_state=80;
        elseif E(j)=='L' && (T(j)>warmwater && T(j)<(warmwater+50))
            disp(sprintf('At %d:%d someone entered, may have taken tea with water that was already warm and left',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
            current_state=0;
        elseif DW_ON==1 && E(j)=='w'
            current_state=30;
        elseif DW_ON==0 && E(j)=='w'&& (T(j)>dishw && T(j)<dishw+50)
            current_state=40;
```

Figure B.12: Patterndetection.m

```
    elseif DW_ON==0 && E(j)=='s'&& (T(j)>dishw && T(j)<dishw+50)
        current_state=41;
    elseif lunch_time==1 && E(j)=='m'
        current_state=50;
    elseif lunch_time==1 && (E(j)=='f' || E(j)=='g')
        current_state=51;
    elseif lunch_time==0 && (E(j)=='f' || E(j)=='g')
        current_state=60;
    elseif lunch_time==0 && E(j)=='m'
        current_state=70;
    end

elseif current_state==11
    if E(j)=='e'
        current_state=12;
    end

elseif current_state==12
    if E(j)=='L'
        disp(sprintf('At %d:%d someone entered, took a coffee and left',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    elseif E(j)=='f' || E(j)=='g'
        current_state=13;
    end

elseif current_state==13
    if E(j)=='g' || E(j)=='f'
        current_state=14;
    end

elseif current_state==14
    if E(j)=='L'
        disp(sprintf('At %d:%d someone entered, took a coffee with milk and left',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    elseif E(j)=='m'
        current_state=15;
    end

elseif current_state==15
    if E(j)=='i'
        current_state=16;
    end

elseif current_state==16
    if E(j)=='L'
        disp(sprintf('At %d:%d someone entered, took a coffee with warm milk and left',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    end

elseif current_state==20
    if E(j)=='v'
        current_state=21;
    elseif E(j)=='L'
        disp(sprintf('At %d:%d someone entered, switched the water boiler on and left',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    end

elseif current_state==21
    if E(j)=='L'
        disp(sprintf('At %d:%d someone entered, heated water, took tea and left',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    end

elseif current_state==30
    if E(j)=='c'
        current_state=31;
```

Figure B.13: Patterndetection.m

```
    elseif E(j)=='b'
        current_state=33;
    end
elseif current_state==31
    if E(j)=='e'
        current_state=32;
    end

elseif current_state==32
    if E(j)=='L'
        disp(sprintf('At %d:%d someone entered, washed his cup in the sink because the dishwasher was working \n and took a coffee',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    end

elseif current_state==33
    if E(j)=='v'
        current_state=34;
    end

elseif current_state==34
    if E(j)=='L'
        disp(sprintf('At %d:%d someone entered, washed his cup in the sink because the dishwasher was working \n and took a tea',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    end

elseif current_state==40
    if E(j)=='s'
        current_state=41;
    end

elseif current_state==41
    if E(j)=='L'
        disp(sprintf('At %d:%d someone entered when the dishwasher had just finished and put his cup in the sink \n instead of emptying the dishwasher and putting it there',fix(T(j)/100),ro
        current_state=0;
    end

elseif current_state==50
    if E(j)=='i'
        current_state=53;
    end

elseif current_state==51
    if E(j)=='g' || E(j)=='f'
        current_state=52;
    end

elseif current_state==52
    if E(j)=='m'
        current_state=50;
    elseif E(j)=='c'
        current_state=58;
    end

elseif current_state==53
    if E(j)=='L'
        disp(sprintf('At %d:%d someone warmed his food in the microwave and had lunch',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    elseif E(j)=='c'
        current_state=54;
    elseif E(j)=='b'
        current_state=56;
    end

 elseif current_state==54
     if E(j)=='e'
         current_state=55;
     end
```

Figure B.14: Patterndetection.m

```
elseif current_state==55;
    if E(j)=='L'
        disp(sprintf('At %d:%d someone warmed his food in the microwave and had lunch and a coffee',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    end

elseif current_state==56
     if E(j)=='v'
         current_state=57;
     end

elseif current_state==57;
    if E(j)=='L'
        disp(sprintf('At %d:%d someone warmed his food in the microwave and had lunch and a tea',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    end

elseif current_state==58
    if E(j)=='e'
        current_state=59;
    end

elseif current_state==59
    if E(j)=='L'
        disp(sprintf('At %d:%d someone took a coffee with milk',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    end

elseif current_state==60
    if E(j)=='g' || E(j)=='f'
        current_state=61;
    end

elseif current_state==61
    if E(j)=='m'
        current_state=62;
    elseif E(j)=='L'
        disp(sprintf('At %d:%d someone entered, opened the fridge and left',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    end

elseif current_state==62
    if E(j)=='i'
        current_state=63;
    elseif E(j)=='L'
        disp(sprintf('At %d:%d someone entered, took something from the fridge, warmed it in the microwave and left',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    end

elseif current_state==63
    if E(j)=='c'
        current_state=64;
    end

elseif current_state==64
    if E(j)=='e'
        current_state=65;
    end

elseif current_state==65
    if E(j)=='L'
        disp(sprintf('At %d:%d someone took a coffee with milk that he had warmed in the microwave',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
        current_state=0;
    end
```

Figure B.15: Patterndetection.m

```
    elseif current_state==70
        if E(j)=='i'
            current_state=71;
        end

    elseif current_state==71
        if E(j)=='L'
            disp(sprintf('At %d:%d someone entered, used the microwave and left',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
            current_state=0;
        end

    elseif current_state==80
        if E(j)=='L'
            disp(sprintf('At %d:%d someone entered, switched the dishwasher on and left',fix(T(j)/100),round(0.6*(fix(T(j))-100*fix(T(j)/100)))));
            current_state=0;
        end

    end

    if E(j)=='L'
        current_state=0;
    end
end
end
```

Figure B.16: Patterndetection.m

Appendix C

Data availability

All data collected is frely available to the public for research purposes. Contact Oriol Prats [oriolpv@kth.se] or Jose Araujo [araujo@kth.se] for it.