Visualisation of MOVEMENT with RSSI

# Introduction

We are working together with 3 different master students: Nick Verbaendert, David Henderickx en Wong Man Hung. Our task is to implement RSSI to make a mobile node to move into the direction of a node which signals an certain event. So we have one mobile node and 3 anchor nodes to help us to go in the right direction. We also need to make our application compatible with the visualization software of Nick and David. We have implemented the following:

* 4 sensors: voltage, temperature, light intensity and humidity
* RSSI
* AM receiver
* Changed the memory allocation (pool & queue)
* Collection Tree Protocol

# header file

|  |
| --- |
| typedef nx\_struct oscilloscope {  nx\_am\_addr\_t moteId;  nx\_uint16\_t count;  nx\_uint16\_t quality;  nx\_am\_addr\_t parentId;  nx\_uint8\_t reply;    nx\_uint16\_t Voltreading;  nx\_uint16\_t Lightreading;  nx\_uint16\_t Tempreading;  nx\_uint16\_t Humidityreading;  nx\_uint16\_t RSSI;    } oscilloscope\_t; |

In the header file we added 5 new elements:

* Voltreading: contains the internal voltage
* Lightreading: contains the light intensity
* Tempreading: contains the temperature
* Humidityreading: contains the humidity
* RSSI: Received Signal Strength Indication

# Configuration file

|  |
| --- |
| components new DemoSensorC() as SensorVoltage; // internal Voltage  components new HamamatsuS1087ParC() as SensorLight; // light intensity  components new SensirionSht11C() as SensorTempHumidity; // temperature and humidity  MultihopOscilloscopeC.ReadVoltage -> SensorVoltage;  MultihopOscilloscopeC.ReadLight -> SensorLight;  MultihopOscilloscopeC.ReadTemperature -> SensorTempHumidity.Temperature;  MultihopOscilloscopeC.ReadHumidity -> SensorTempHumidity.Humidity; |

In this file we add the components for the 4 different sensors. We connect them to the read interfaces so that it is usable in the main program.

|  |
| --- |
| components new AMReceiverC(AM\_OSCILLOSCOPE) as MobileReceiver;    MultihopOscilloscopeC.MobileReceive -> MobileReceiver; |

To locate a mobile node in a network we take the RSSI from 3 differents nodes and with those values we can calculate where the mobile node is situated. Thus, we add the component AMReceiver to listen to the packets sent by the other 3 nodes to get the RSSI.

We connect the AMReceiverC to the Receive interface.

|  |
| --- |
| components CC2420ActiveMessageC as Radio;  MultihopOscilloscopeC -> Radio.CC2420Packet; |

We add the component CC2420ActiveMessageC (CC2420 is the radio) to be able to get the RSSI.

|  |
| --- |
| components CtpP as CollectP;  MultihopOscilloscopeC.CollectInfo -> CollectP; |

We add the CtpP to get some more information like the parentID and expected transmissions (ETX). CTP uses expected transmissions as its routing gradient.

# module file

## module part

In the module part we declare the interface which we are going to use and the commands included.

|  |
| --- |
| interface CC2420Packet; |

Interface for getting RSSI

|  |
| --- |
| interface Receive as MobileReceive; |

Interface voor de AM receive.

|  |
| --- |
| interface CtpInfo as CollectInfo |

Interface for getting certain data from CTP like quality and parentID.

|  |
| --- |
| interface Queue<message\_t \*> as RadioQueue;  interface Pool<message\_t> as RadioMessagePool; |

Interface for the use of a queue and pool. This is used for storing received messages for the anchor nodes.

|  |
| --- |
| interface Read<uint16\_t> as ReadVoltage;  interface Read<uint16\_t> as ReadLight;  interface Read<uint16\_t> as ReadTemperature;  interface Read<uint16\_t> as ReadHumidity; |

Read interface reads out the different sensors

## implementation part

|  |
| --- |
| bool voltageIsRead = FALSE;  bool lightIsRead = FALSE;  bool tempIsRead = FALSE;  bool humidityIsRead = FALSE;  bool rssiIsRead = FALSE;  bool root = FALSE; |

We add some bools to see when a certain reading is finished and to know which node is the root.

|  |
| --- |
| event void Timer.fired(){  if (call ReadVoltage.read() == SUCCESS) voltageIsRead = TRUE;  if (call ReadLight.read() == SUCCESS) lightIsRead = TRUE;  if (call ReadTemperature.read() == SUCCESS) tempIsRead = TRUE;  if (call ReadHumidity.read() == SUCCESS) humidityIsRead = TRUE;  getRssi(&sendbuf);    if(voltageIsRead && lightIsRead && tempIsRead && humidityIsRead && rssiIsRead){  local.count++;  if(!root)  {  oscilloscope\_t \*out;  if (!sendBusy){  out = (oscilloscope\_t \*)call CollectSend.getPayload(&sendbuf);  memcpy(out, &local, sizeof(oscilloscope\_t));  post collectSendTask();  }  else  {  message\_t \*newmsg = call RadioMessagePool.get();  if (newmsg == NULL)  {  report\_problem();  }    out = (oscilloscope\_t\*)call CollectSend.getPayload(newmsg);  memcpy(out, &local, sizeof(oscilloscope\_t));    if (call RadioQueue.enqueue(newmsg) != SUCCESS)  {  call RadioMessagePool.put(newmsg);  fatal\_problem();  }  }    }  }  } |

Timer.fired is the main event of the program. Here we collect the data and transmit it when bool sendbusy = FALSE. If the bool is false, then we store the data in the right format on the queue.

|  |
| --- |
| task void collectSendTask() {  if (!root) {  if (call CollectSend.send(&sendbuf, sizeof(oscilloscope\_t)) == SUCCESS)  {  oscilloscope\_t \*o;  sendBusy = TRUE;  o = (oscilloscope\_t \*)call CollectSend.getPayload(&sendbuf);  printf("De id = %u\n",o->moteId);  call PrintfFlush.flush();  }  else  report\_problem();  }  } |

CollectSendTask is a task and is called on to transmit his own data and data received from the anchor nodes, so it is called on in the events ‘Timer.fired’ and ‘Mobilereceive.receive’. The task is adapted in the way that we don’t need to copy the data into the message\_t sendbuf here.

|  |
| --- |
| event void CollectSend.sendDone(message\_t\* msg, error\_t error){  if (error != SUCCESS)  report\_problem();    sendBusy = FALSE;    if (call RadioQueue.empty() == FALSE)  {  message\_t \*queuemsg = call RadioQueue.dequeue();  if (queuemsg == NULL)  {  fatal\_problem();  return;  }  memcpy(&sendbuf, queuemsg, sizeof(message\_t));  if (call RadioMessagePool.put(queuemsg) != SUCCESS)  {  fatal\_problem();  return;  }  post collectSendTask();    }    local.reply = NO\_REPLY;  report\_sent();  } |

In the Collect.sendDone we put the bool sendBusy to FALSE and check if there is still data on the queue. If so, then we transmit them.

|  |
| --- |
| event void ReadVoltage.readDone(error\_t ok, uint16\_t data) {  if (ok == SUCCESS)  {  fillPacket();  local.Voltreading = data;  }  else  {  data = 0xffff;  report\_problem();  }  }  event void ReadLight.readDone(error\_t ok, uint16\_t data) {  if (ok == SUCCESS)  {  fillPacket();  local.Lightreading = data;  }  else  {  data = 0xffff;  report\_problem();  }  }    event void ReadTemperature.readDone(error\_t ok, uint16\_t data) {  if (ok == SUCCESS)  {  fillPacket();  local.Tempreading = data;  }  else  {  data = 0xffff;  report\_problem();  }    }    event void ReadHumidity.readDone(error\_t ok, uint16\_t data) {  if (ok == SUCCESS)  {  fillPacket();  local.Humidityreading = data;  }  else  {  data = 0xffff;  report\_problem();  }    }    uint16\_t getRssi(message\_t \*msg){  local.RSSI = (uint16\_t) call CC2420Packet.getRssi(msg);  rssiIsRead = TRUE;  return local.RSSI;  } |

In the above events, we fill our packet with the data it needs.

|  |
| --- |
| void fillPacket() {  uint16\_t tmp;  call CollectInfo.getEtx(&tmp);  local.quality = tmp;  call CollectInfo.getParent(&tmp);  local.parentId = tmp;  } |

The function fillPacket further fills the packet we want to transmit with certain data like quality and parentID. This function is called on when we have read the different sensors.

|  |
| --- |
| event message\_t\* MobileReceive.receive(message\_t\* msg, void\* payload, uint8\_t len) {    oscilloscope\_t \*in = (oscilloscope\_t\*)payload;  oscilloscope\_t \*out;    report\_received();  if(!sendBusy)  {  out = (oscilloscope\_t \*)call CollectSend.getPayload(&sendbuf);  if (len != sizeof(oscilloscope\_t))  {  return msg;  }  else  {  memcpy(out, in, sizeof(oscilloscope\_t));  }    post collectSendTask();  }  else  {  message\_t \*newmsg = call RadioMessagePool.get();  if (newmsg == NULL)  {  report\_problem();  return msg;  }    out = (oscilloscope\_t\*)call CollectSend.getPayload(newmsg);  memcpy(out, in, sizeof(oscilloscope\_t));    if (call RadioQueue.enqueue(newmsg) != SUCCESS)  {  call RadioMessagePool.put(newmsg);  fatal\_problem();  return msg;  }  }  sendBusy = TRUE;  return msg;  } |

The Mobilereceive is the next main part of the application. In this event we receive the different packets transmitted by the anchor nodes. We take the payload from these packages and transmit them if the bool sendBusy = FALSE. If not then we put the data on a queue in the right format.