PhyNetLab

Ultra-low power learning

Introduction:

A world full of sensors leads to two main developments: Gigantic centralized services, gathering and analysing data, and highly distributed data generation. The latter demands respecting a multitude of restricted resources: Computational power, limited communication speed and reliability, often limited energy. The PhyNodes are embedded computing and sensing platforms developed at the collaborative research centre as a large-scale testbed for the Internet of Things.

Hackathon:

Team of 3-5 comprising of participants with interests in embedded programming and Machine learning are created.
KRATOS and its embedded systems are programmed by one part of the team. Toolchain setup, Board bring up and accessing sensors from the PhyNode are tasked to this team.

The other part of the team will be working on developing a machine learning algorithm. This machine learning algorithm will be developed with the help of the given dataset. Please find the data in the following link:

`http://phynetlab.com/dataset.zip`

After validation and verification of the algorithm, the model should be exported and programmed in the PhyNodes. This means, using the dataset provided, the learned goal must be proved in the ultra-low power system (PhyNode).

The evaluation of the algorithm depends on two factors. The accuracy of the goal estimation and the amount of energy used. There could be a tradeoff in producing accurate estimations using more energy and this must be decided during the development phase to strike the best balance for goal estimation.

Teams tasks:

- Programming PhyNode
  - Setting up the toolchain.
  - Using KRATOS for sensor data acquisition and data collection.
  - Programming the model learned out of the data set.
- Machine learning
  - Understand the data provided to estimate the goal
  - Validate that the model, if it works well for the given data set
  - Export the model into the PhyNode
- Program the PhyNodes with the learnt model and validate on board.

Organizers:

KRATOS Toolchain: Markus Buschoff
Programming API: David Tondorf, Mojtaba MasoudiNejad

Machine Learning: Jens Buß, Andreas Pauly, Stefan Michaelis

Feel free to ask your questions to anyone and you will be directed to the right person.

Step 1: Hello world

30 – 45 mins

Tasks:

- Setup your toolchain
  - Use the bootable USB or
  - Use the Virtual machine image provided (ask the organisers).
- The KRATOS toolchain is already setup.
- Use the getting started app for steps 1 to 3.

Tools used:

Serial output via UART:

```
#include "drivers/eUSCI_A/uart/prototype_uart.h"
```

After including the necessary header file (see above) you can just use the global object named "uart" to output data using the cout syntax like you would in standard c++.

Example:

```
int i = 42;
uart << "This text will be output via UART" << endl;
uart << "This is a number: " << i << endl;
uart << "This is a hex number: " << hex << i << dec << endl;
```

LED

The PhyNode has one LED that is connected to port P4.2 (port 4, pin 2)

To use it you first have to set the LED pin to output mode and then turn it on/off by setting the pin high/low. The easiest way to do this is to use the macros from

```
"drivers/gpio.h":
setOutput(port,pin),
pinHigh(port,pin)
pinLow(port,pin)
pinToggle(port, pin)
```

Waiting/Sleeping

To put the microcontroller (MCU) to sleep you need a Guarded_Buzzer or Guarded_RTCBuzzer object and call it's sleep function.

A Guarded_Buzzer can be used to sleep short periods of time (seconds) with millisecond resolution.

If you want to put the MCU to sleep for longer periods (30 seconds or so) you can use a Guarded_RTCBuzzer, which has a resolution of seconds.

Example:

```
#include "syscall/guarded_buzzer.h"
#include "syscall/guarded_rtcbuzzer.h"

Guarded_Buzzer buz;
buz.sleep(100);   // Sleep for 100ms

Guarded_RTCBuzzer rtc;
rtc.sleep(42);    // Sleep for 42 seconds
```

Goal:

- Print “Hello World!” in UART
- Blink the LED in the PhyNode

Step 2: Data acquisition

30 – 45 mins

Task:

There are 3 values you can get from the sensors in the Phynode:

- Light intensity
- Acceleration in x, y, z-direction
- Temperature

Use the API provided and print the output to the serial port.

Turning off the sensor after using them could be good practice to save energy for the hackathon task.

Tools used:

Light sensor:

Relevant functions:

Turn on the light sensor and wait for a second until the sensor is initialized and has sampled the light values.

```
/**
 * Returns the current light intensity in lux as reported by the MAX44009
light sensor.
 */
uint16_t MAX44009::getLux()
/**
 * Turn on power to light sensor. Call this before using the sensor.
 */
void MAX44009::powerOn();
/**
```

```
 * Turn off light sensor power.
 */
void MAX44009::powerOff();
```

Usage:

```
// Already included in Summerschool.h
#include "drivers/lightsensor/max44009.h"

// lightsensor is a global MAX44009 object declared in max44009.h
lightsensor.powerOn();
uint16_t lux = lightsensor.getLux();
lightsensor.powerOff();
```

Acceleration:

Relevant functions:

```
/**
 * Returns a struct containing the current acceleration in x-, y- and z-
direction.
 * The struct is defined as:
 * struct mems_xyz_data
 * {
 *     int16_t xdata, ydata, zdata;
 * };
 */
struct mems_xyz_data MEMS_Accelerometer::get_xyz();
/**
 * Powers up the mems sensor and initializes it.
 * Has to be called before the sensor can be used.
 */
void MEMS_Accelerometer::init();
/**
 * Turns off power to the mems sensor.Can be used to conserve energy when
the sensor is not used.
 * Remember to call init() again if you want to use it again.
 */
void MEMS_Accelerometer::powerOff();
```

Usage:

```
// Already included in Summerschool.h
#include "drivers/accelerometer/mems.h"
// mems is a global MEMS_Accelerometer object declared in mems.h
mems.init();
mems_xyz_data xyz = mems.get_xyz();
mems.powerOff();
```

Temperature:

Temperature is measured by the same sensor that measures acceleration, to the same `init()` and `powerOff()` functions are relevant here as well. You need to set the offset value on the PhyNode because the sensor is not calibrated by default. You can get the value for your PhyNode from nodeData.h, which is in the same folder as the code for the app.

Relevant functions:

```
/**
```

```
 * Returns the current temperature in Celsius * 100 (e.g. 24.23°C would be
represented as 2432)
 */
int16_t MEMS_Accelerometer::get_temperature();
/**
 * Sets the temperature offset value
 * You can get the value for your PhyNode from nodeData.h, which is in the
same folder as the code for the app (e.g. GettingStarted app for the first
3 exercises)
 */
void MEMS_Accelerometer::set_temperature_offset(int16_t offset);
```

Usage:

```
// Already included in Summerschool.h
#include "drivers/accelerometer/mems.h"
mems.init();
int16_t temperature = mems.get_temperature();
mems.powerOff();
```

Goal:

- Read all sensor data which will be later used in the localization.
  - Light intensity MAX44009
  - Accelerometer & Temperature ADXL362
- Output them via UART

Step 3: Ping – Pong

30 – 45 mins

Task:

Sending the ping pong and communicating with the router. Even though this part will be handled by the summerschool app automatically, it is important to know the energy leading energy factor due to wireless communication.

Tools used:

The PhyNode uses a Texas Instruments CC1200 radio chip for communication in the 868 MHz band.

Radio: CC1200 Transceiver

Relevant functions:

```
/**
 * Power up radio chip
 */
void cc1200::powerOn();

/**
 * Power down radio chip
 */
void cc1200::powerOff();

/**
 * Initialize radio chip. This must be called after powerUp() to set the
correct frequency and other settings
 */
void cc1200::init();
```

```
/**
  * Set the physical address of the radio chip. You will need to set
this address to receive unicast messages from the gateway (e.g. PONG).
  * Use the id of your PhyNode as the address.
  */
void cc1200::set_address(uint8_t address);

/**

/**
  * Set the physical address of the radio chip. You will need to set
this address to receive unicast messages from the gateway (e.g. PONG).
  * Use the id of your PhyNode as the address.
  */
void cc1200::set_address(uint8_t address);

 * Put radio chip in receive mode. Has to be called before and packets can
be received.
 * This uses a lot of power.
 */
void cc1200::listen();

/**
 * Send a radio packet.
 * A CCPhyPacket is defined as follows:
 * typedef struct __attribute__((packed)) {
 * unsigned char statusbyte1; // Contains RSSI
 * unsigned char statusbyte2; // BIT 7 CRC_OK, BIT 6:0 LQI
                                  (Link Quality Indicator)
 * unsigned char length; // Payload length only;
 * unsigned char address; // PHY Address, use 0xAA to send to Gateway, 0x00
to broadcast (but don't broadcast, you only need to communicate with the
gateway)
 * unsigned char payload[128];            // Payload
 *   } CCPHYPacket;
 *
 * Since a packet is very large and RAM/stack is limited, don't create
packets in RAM.
 * You can use the global packets "packet" and "ack" defined in
Summerschool.h and SummerschoolGettingStarted.h for all communications
 */
void cc1200::send(CCPHYPacket& pkg);

/**
 * Blocks until a packet is received, listen() has to be called first to
put the radio in RX mode.
 */
void cc1200::receive(CCPHYPacket& packet);
```

Usage:

For usage examples, best have a look at `sendResponse()` in Summerschool.cc

Access point beacon/heartbeat protocol

- All 3 gateways will send out beacons periodically.
- The order is always beacon 1 --> beacon 2 --> beacon 3.
- A beacon always consists of the payload "BEACON_n", where n is the number of the gateway that sent the beacon.
- So, if you receive a packet with payload "BEACON_1", you know that this packet was sent by gateway 1.

- Beacon 1 can also contain the sensor data from the 2 reference boards.
- If it does, the payload length for that packet is `strlen("BEACON_1") + 2*sizeof(DataPacket)`
- (see Summerschool.h for the definition of `DataPacket`) and the payload consists of the regular BEACON string and 2 `DataPacket` containing the data from the reference boards.

Sending a PING

- A ping is a packet with 5 bytes of payload:
- The string "PING" (4 bytes) + the id of the node that send the ping (1 byte).
- The gateway will respond to a PING by sending a PONG (packet with payload "PONG") after roughly 100ms.
- The PONG will only be sent to the node that sent the PING.

Goal:

- Read the RSSI from the beacons. RSSI usually have a very good estimate of the physical distance between the Acces point and PhyNode.
- Send the PING and receive PONG from gateway.

Step 4: Hackathon

Until 27.09.2017 – 22:00

Task:

- Use the data we provide to create a machine learning the model to predict the location of the PhyNode.
- Implement this model in C++ on the PhyNode.
- Summer school app will take care of the node initialization and also the packet assembling
  - There is a timer until when the user thread will be running
  - After the timer is expired, the user thread will be killed by the `summerschoolThread` and the communication with the gateway will happen
  - The user can find the remaining time for the thread using this function call: `summerschoolThread.getRemainingTime()`
  - The return of this function is remaining time in **seconds**
- Set the position you get from your algorithm of the PhyNode using `summerschoolThread.setPosition(uint8 Position)`
- Before you thread is killed, energy accounting is done by KRATOS for the user thread/
  - This value will also be used in the evaluation of the best algorithm.
- Take care about the energy that is being used for your algorithm, the amount of time you poll the sensors and the receive the beacons.

Tools used:

Data description:

Data is collected from a warehouse replica with two side racks with one floor in between.

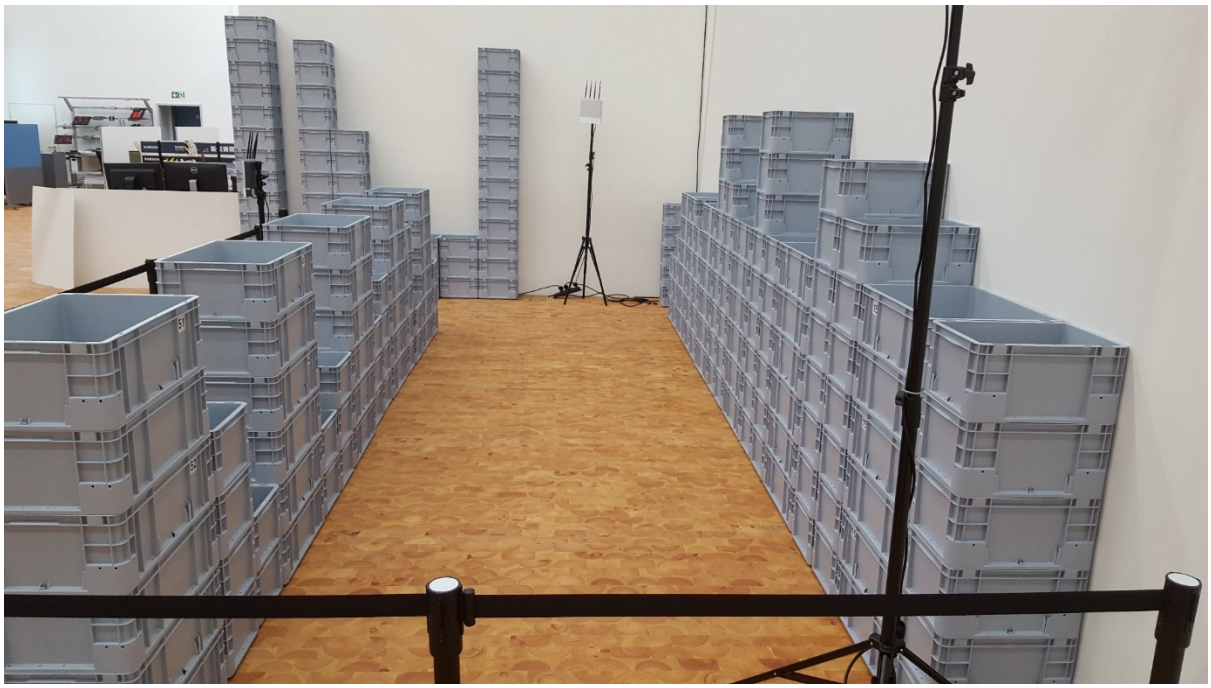

Fig 1: Overview of one floor with two racks as a warehouse replica

The side at right hand of the fig. 1 is named side 1 and left side is the side 2.
At each side, there are 5 columns and three rows in addition to one reference position. The overall numbering of positions is shown in Fig 2 below:

	111		121		131		141		151	Ref1
	112		122		132		142		152	
	113		123		133		143		153	

	211		221		231		241		251	
	212		222		232		242		252	
Ref2	213		223		233		243		253	

Fig 2: Overall numbering of positions

In addition to places there are three access points (AP) that send heartbeat/beacons in a periodic way sequentially after each other.

For each data collection experiment, 20 PhyNodes (numbered from 1 to 20) are randomly spread over the 30 possible places and PhyNode 21 is always at Ref1 while PhyNode 22 is positioned in Ref2.

At receive of each beacon signal, PhyNode stores the RSSI value of the related Access Point. When all three RSSI values are available, PhyNode measures the sensor values and stores all these values as one entry into a .csv data with order as below:

- Time: Human readable time
- UNIX_T: A Unix time value
- RSSI1: Signal strength from the AP1

- RSSI2: Signal strength from the AP2
- RSSI3: Signal strength from the AP3
- Lux: Light intensity in Lux
- Acc_x: Acceleration in x direction (Toward floor)
- Acc_y: Acceleration in y direction (in the direction of corridor)
- Acc_z: Acceleration in z direction (in the direction of human operator)
- Temp: A calibrated temperature value
- ID: ID Number of the PhyNode (1 to 22)
- Pos: Known position entered manually before data collection in the form of side, column, row
- Si: Side number as a separate value
- Co: Column number as a separate value
- Ro: Row number as a separate value

Each entry is sent back to the access point and stored there. PhyNodes provide these information in a periodic way that there should be one entry per second.
Note: Although it had been taken care to avoid missing data, same as any physical system it is possible that one entry for a PhyNode be missed.
Each experiment takes 90 seconds which should provide enough information for learning one specific position with a PhyNode. At the end, each dataset is restored from the AP and can be accessed through the provided .csv data.

Name of each file provide information about the data collection time.
In addition, the lighting condition of the hall during the experiment can be seen after the letter 'B', which is in percentage of the maximum luminosity of the lighting system in the warehouse.

Goal:

- Set the position you get from your algorithm of the PhyNode using `summerschoolThread.setPosition(uint8 Position)`
- The position will be updated after the timer to the access points with energy accounting.
- Here you implemented a machine learning algorithm in an ultra-low power wireless sensor node and predicted its position using meta information available on the sensors.