Software Aspects of Embedded Systems

Real-time Synchronization on LocoKit

Supervisor Ulrik Pagh Schultz

Oscar Adrian Diaz del Rosario
Simon Sebastian Stiil Frederiksen
Mats Larsen

Deadline: d. 07.01 2013

SDU - Mærsk McKinney Møller inst.

Contents

1 Introduction

This report is about a synchronization project which aim to synchronize two locokit boards over a wireless network and test the synchronization by controlling locokit motors. This include to produces a synchronization algorithm to work in a wireless environment and motor controlling. This feature is essential in sensor networks where the time is an important factor in the measurements e.g. position and velocity. In those cases a common clock is not a possibility, therefore this method is used to solve the problem, on an alternative way. We choose this project, because we think it is relevant according to our course in EMB4. In this project we tried to synchronize a locokit, therefore our challenge in this project were to learn about locokit and get locokit's wireless and motors to work properly. Another challenge was to understand, how wireless synchronization in sensor networks are working and to choose which one how fits best to our project. Here we used the basic of the RBS to make the synchronization. When the time is synchronized, a test setup is made to illustrate, how well the synchronization worked.

2 Wifi Networking

In order to get the two devices to communicate with each other we need to set up a wifi communication between the two devices. If we only want the two devices to communicate with each other we can set up what is called an Ad-hoc network. But as we want to set up LCM (Lightweight Communications and Marshalling) we want to be able to monitor the traffic between the nodes. This can only be achieved if the PC is also on the network requiring the network to be an infrastructure network. This is a network where a wireless router or access-point bridges a regular network or internet connection to wireless devices on the network. This can be seen on figure 2.1.

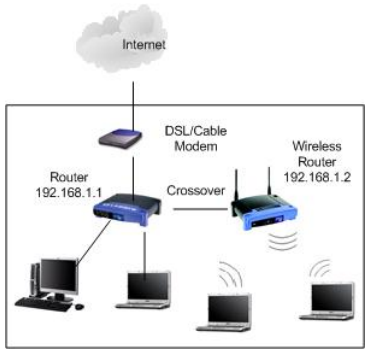

Figure 2.1: Wireless router as access point

Figure 2.1 is a combination od wired and wireless network. Wireless networking is no problem to set up in windows but when setting it up in linux from command line it's a little more complicated we use the two configuration files /etc/networking/interfaces and /etc/wpa supplicant.conf. The first file is for setting up ip configuration general wifi configuration (WPA/WEB/no encryption) and the second file if we use WPA for the additional layer of security that is put besides the regular configuration.

Content of /etc/networking/interfaces can be seen in figure 2.2.

```
1 allow-hotplug wlan0
2 iface wlan0 inet dhcp
3       pre-up wpa\_supplicant -Dwext -iwlan0 -c/etc/wpa\_supplicant.conf -B
4       down killall wpa\_supplicant
```

Figure 2.2: Code wireless configuration of /etc/networking/interfaces

This configuration starts the wpa_suplicant program using the /etc/wpa supplicant.conf before starting the network adapter. Then when stopping it will kill the program.

Content of /etc/wpa supplicant.conf can be seen in figure 2.3.

Further information about setting up wifi on the boards is available in the midterm report. After this setup the two nodes are available through SSH, SCP, HTTP and other protocols as well. Using the command ifconfig we are able to get the ip address of the devices.

```
root@ubuntu:~# wpa_passphrase "SWEAT WLAN" sweat4all
network={
        ssid="SWEAT WLAN"
        #psk="sweat4all"
```

Figure 2.3: Code wireless configuration of /etc/wpa supplicant.conf

3 Synchronization algorithm

This topic was been an interesting topic during the last 10 years. Many protocols have been implemented during to for-fill the need for a sensor networks e.g. to detect movement and location. A description of the basic will be explained and then one of them are selected there fit best to our problem.

3-1 Motivation

Synchronization is useful or sometimes required in many applications because knowing the time is often an essential factor in all systems, especially in networks. Examples can be, see list below.

- Coordination of wake-up and sleeping times, to optimize energy efficiency
- Ordering of collected sensor data and events
- Co-operation of multiple sensor nodes
- Estimation of position information

To achieve that, we need a synchronization algorithm to compensate of offset between clocks and compensate drift between clocks. This is needed because hardware clocks are not perfect. There will always be variations in oscillators. To compensate for drift the synchronization has to be periodic. Concerning clock drift, there will be a random deviation from the nominal rate dependent on power and temperature.

3-2 Wireless Network Synchronization

There is a huge differences to implement synchronization algorithm for wired and wireless networks. The two most common algorithms for wired networks are Network Time Protocol(internal synchronization) and Global Positioning System(external synchronization). It can be either internal and external. Those traditional protocols can't be applied in wireless sensor networks, because nodes in sensor networks are often battery powered, therefore it costs to much power and processor power and memory are also limited. It can't be to complex.

Synchronization in a sensor network does not mean that all clocks are exactly matched and also precise clock synchronization is not always important. It depends of the application, sometimes the design can be very simple.

There exist three basic types of wireless synchronization.

- Happens-before
- Local synchronization
- Global synchronization

Happens-before is the simplest one. It isn't related to the real-time. Instead it keep track of each other's events. Here will all the processes don't care about the exact time, but more about which event there happened first. There will be a relation between these events in order to roughly synchronize the clocks.

Local synchronization will keep track of offset and drifts in e.g. a hop, then has the ability to synchronize their local clocks with another local clock.

Global synchronization is like GPS, that each node have a common UTC-time to synchronize after.

Local synchronization is decided to be used, because a precise clock synchronization is required, and global information is not available in our project.

3-3 Local synchronization

There exist basically just two methods..

- Sender/Receiver Synchronization
- Receiver/Receiver Synchronization

The deterministic problem concerning wireless is larger problem then wired networks, because the exact time during transmitting and receiving a message is more critical to predict for wireless networks. To optimizes the accuracy of the synchronization, this deterministic problem has to be minimized. In the process to transmit and receive there will be some jitter or more precisely four delay components, see the figure 3.1.

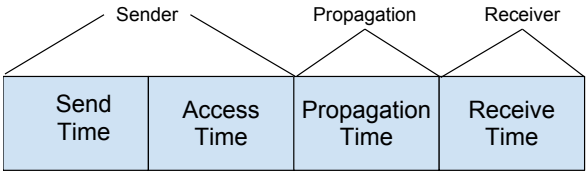

Figure 3.1: Four delay components

These delay components are send time, access time, propagation time and receive time. The send time is the time to construct a message to transmit. The access time is the MAC-layer delay to accessing the network. e.g. waiting in a TDMA protocol. The propagation delay is that time, it takes for the bits to be physically transmitted. The receive time is that time for the receiving node to transport the message to the application layer and then process it. The problem is that the delay exist but also how to predict the delay. There exist some technique to reduce and also eliminate some components.

3-3-1 Sender/Receiver Synchronization

It is a round-Trip-Time based synchronization. The receiver synchronizes itself with the sender. With this approach the delay and offset can be determined. There exist different approaches.

Time-sync Protocol for Sensor Networks (TPSN)

It uses a tree to organize the network topology. It consists of two phases, a level discovery phase and a synchronization phase.The level discovery make a hierarchical topology of the network by using Breadth-first-search. Here will all nodes have a level. According to the synchronization phase all level i nodes will be synchronize with nodes in level i-1 in a two-ways communication. Here will the root be in level $i = 0$.

Flooding Time Synchronization Protocol (FTSP)

It is similar to TPSN, but it has some improvements respectively to TPSN. It still has a root node, which informs the next level nodes with a message containing of the global time. Receivers record the local time stamp for the received message to determine the offset. It also uses a linear regression to compensate for drift. This is designed for large multihop networks. The root is selected dynamically and reselected periodic to ensure to keep the global time of the network and resist of changing in the network topology.

3-3-2 Receiver/Receiver Synchronization

Here is looked at the Reference Broadcast Synchronization(RBS). This principle is that a beacon node will broadcast a message to all receivers in a hop. This beacon message doesn't contain any useful information. It just activate receivers to exchange their time with each other to determine the offset. This receiver comparison is also broadcasting. The time they use to compare is that time each receiver get by recording the timestamp when a receiver received a beacon message. For multihop networks more beacon messages have to be send to activate the other hops. Advantage of RBS is that the critical path is reduced by negligible the send time and access time components, which is the sender. This is illustrated in figure 3.2. Here shows that the deterministic problem is minimized because the the uncertainty in the sender is eliminated.

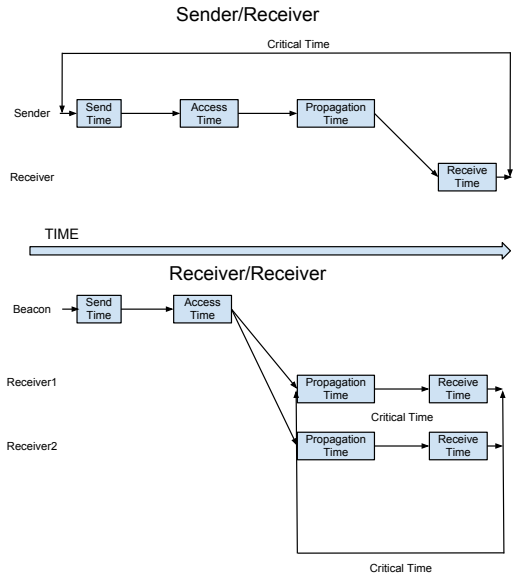

Figure 3.2: Illustrate the difference between sender/receiver and receiver/receiver critical paths

If the range in the hop is very small the propagation time is stable and minimum. If this assuming is used, the only unpredictable turns out to be the receive time.

3-4 The selected algorithm

RBS is decided to be implement in our project. The reason is that only one hop is required and by the way these nodes have a small distances between them. So all this advantages that the sender/receiver gives by large scale multihob is not useful here. The major advantage by using RBS is that the sender is eliminated. Each node will be informed by all other nodes' timestamps. With those facts a node can determine the offset to a respectively node by take the difference between them $offset = neighbour_time - current_time$. When the synchronization is periodic, it will give more observations, to perform a averaging of the offsets, which will be more appropriately.

3-5 Clock drift compensation

It is not possible to produce oscillators that oscillate at the same rate, this is the reason that the clock will drift over time. This behaviour will affect the offset, and the mean offset doesn't take this into account. Least square linear regression(lslr) will do the job, to find the best fit line through offset over time. Here is made a estimation of a line by estimators of the slope and intercept, by this equation $Y = A + Bx_i$. Below are the equations to compute A and B.

$$B = \frac{\sum x_i Y_i - \overline{x} \sum Y_i}{\sum x_i^2 - n\overline{x}^2} \tag{3.1}$$

$$A = \overline{Y} - B\overline{x} \tag{3.2}$$

The estimate will be calculated as this

$$\hat{Y} = A + Bx_i \tag{3.3}$$

4 Motor control

This contains two problems. First dealing with the problem of distributing wanted position of the motor over WIFI between the controllers and then making the motors to follow the wanted position of the motor.

4-1 Distributing motor position

We looked at a few different types of solutions in order to distribute the wanted motor position at first we read the motor position on one motor and copying this as destination to the other motor (A), see figure 4.1. Then the solution of having a master controller that sent the position to both controllers at the same time giving the same time delay (B), see figure 4.2, and finally a solution of using the RTC (Real Time Clock) to calculate the wanted position on all of the devices (C), see figure 4.3.

4-1-1 Mirrored control (A)

The advantage of this control is that it is easy to read the position from the Loco kit motor and then transfer the position to the other motor. Disadvantage is the time delay will make it easy to see the difference between these two motors movement. There is a 150-250ms ping delay between the two controllers (time for a message to be sent back and forth between the motors). So this solution is discarded right away.

4-1-2 Master Controller (B)

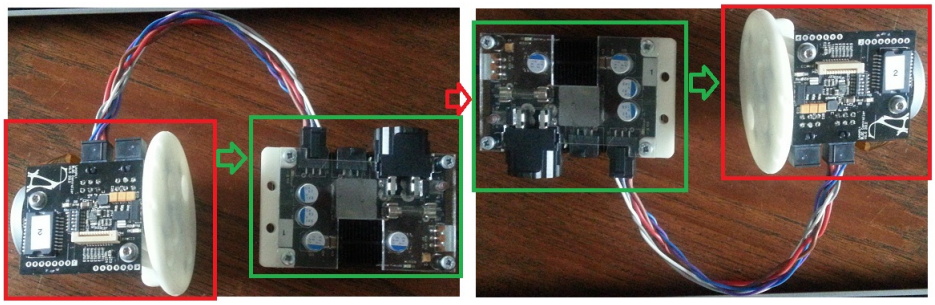

Figure 4.1: Master controll

In this solution a controller is setup to send the wanted location to both motor controllers at the same time giving the ping delay no impact as it will be the same for all units. It leaved the disadvantage of having to use a extra PC or controller to send the location to all the active units. This is a possible solution.

4-1-3 Distributed Control (C)

For distributed control we use the Reference Broadcast Synchronization to have an identical RTC on both controllers. We can then use the RTC to make the same simulated motor position on both units

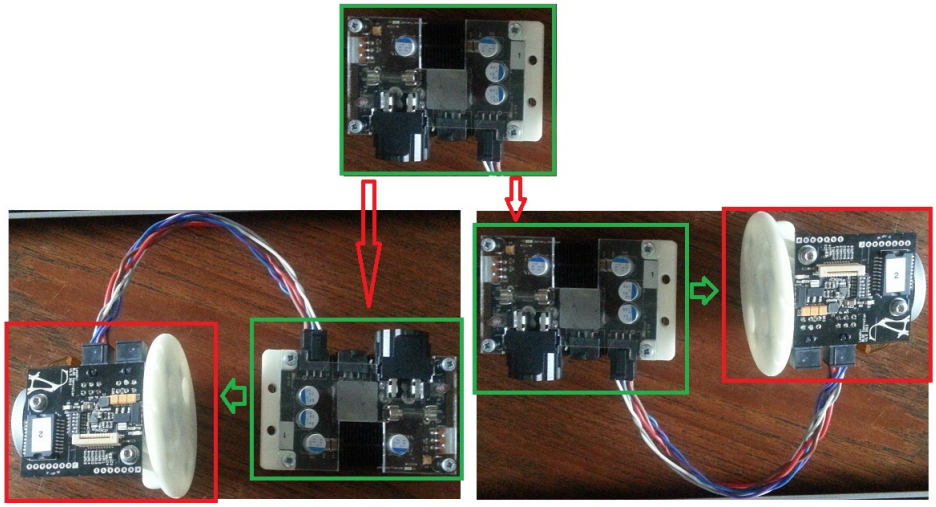

Figure 4.2: A and B motor control

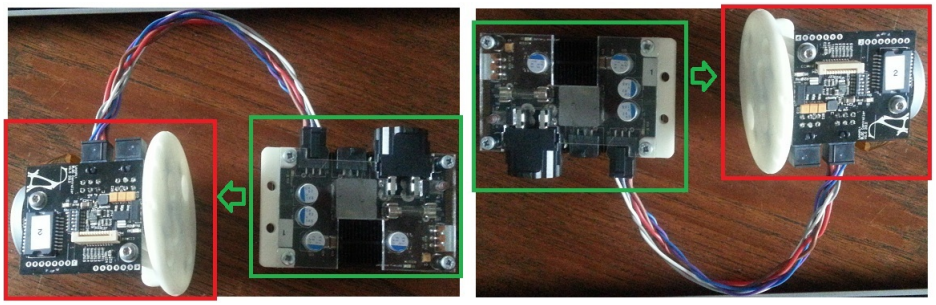

Figure 4.3: Distributed control

and try to mimic the location. This solution is of great interest as it uses less hardware and gives us higher precession then any of the other solutions.

5 Implementation

This chapter will briefly consist of the basic implementation in our project.

5-1 Lightweight Communications and Marshalling

LCM has been the communication tool used for message passing and marshalling between the different nodes. It is a set of libraries and tools and provides a publish/subscribe message passing model and automatic marshalling/unmarshalling code generation with bindings for applications in a variety of programming languages. Some of its main features are:

- Low-latency inter-process communication
- Efficient broadcast mechanism using UDP Multicast
- Essentially unlimited packet size
- Supports C, C++, Java, Python, MATLAB, and C#

The setup of LCM on the different devices (PC and Gumstix boards) was done by following the instructions from http://code.google.com/p/lcm/wiki/BuildInstructions, which gives a detailed information of how to compile and install it. It gives a serie of required packages and the desired LCM release which need to be installed first, and a tutorial to get started with this enviroment. This tutorial gives a walkthrough the main tasks for exchanging messages between two applications, and gives the enough information to create/use the first examples. It contains:

1. Create a type definition
2. Initialize LCM in the application
3. Publish a message
4. Subscribe to a channel and receive a message

Once a lcm file is created, the `lcm-gen` is the tool in charge of generating the bindings for the specific language. Python was the first language used for working with LCM, but due some package problems while installing it in one of the boards, C was used instead. After generating these language-specific bindings, some examples are created which can be run to subscribe to a default channel and send and listen depending on what is chosen.

Different files were created for the purpose of the project and are explained in the next paragraph, and can be found on the folder attached to this document.

5-2 RBS

There is implement a beacon node named beacon_reference, which inform all receivers to begin the synchronization process. Two receivers are implement, named node_sync_1 and node_sync_2. Basically several receivers could be ease added, it is just the node number there distinguish them. In the node there will be two threads, one for the synchronization process and one for the motor controlling. The program will set-up several things including LCM. It means when it receives a message over LCM on a specific channel it will call the method my_handler, which also include executing of RBS.

The RBS is implemented as a state machine to handle beacon messages and receiver to receiver messages. Beacon state has this flow of activities.

1. Obtain localtime
2. Construct a message with localtime
3. Publish this message, to all receivers

Receiver to receiver state has this flow of activities.

1. Store neighbour node's localtime and offset between neighbour nodes and current node
2. Compute mean of offset and store it
3. Compute least square linear regression and store it
4. Handle linked list

The thing here is how to store this information. There is some among of data, so it has to implement in a reasonable way. Here structs are used. It is implemented as a linked list, so a integer is running in loop which point on the newest element, and there is also a pointer though the whole list. The important thing here is that the list has a specific number specifying the length. When the list is full it will not shift all elements, which is heavy operation. Instead it replace the oldest element with the newest and change front and end pointer, which is more effectively. The design of the struct can be seen in 5.1. RBS_INFO is the top struct, which reference to all NODES who are in the network. A specific node reference to NODE_INFO which contain all stored values.

5-3 Controlling the Locokit Motors

The Locokit API contains a wide variety of motor control functions. Most of these functions are though related to a single controller having several motors connected rather the multiply controllers talking to each other about motor control. So we have decided to create our own PID controller using functions for setting motor speed and getting motor position. We spoke to Rasmus Petersen and he gave us a quick introduction to the Locokit motors. How to get information from them and how to use them for control functions. Using the Locokit API requires first importing the locoapi.h into our C program and then afterwards referring to a "actuateMotors-settings.cfg" file when using the initializeLocoAPI (char *setup_file) this file needs to contain information about what functionalities the LOCOkit API have available. The problem about the setup file is that there is little documentation about how to use it. There is a couple of example files that contain also very little information. So after a lot of trial and especially error a file with the content below was created.

- First motor controller
- Actuators (used id 0 to enable broadcast)
  - ARM7MC = 0,2

```
struct NODE_INFO  // store data for each TRIAL
{
  struct NODE_INFO *next;   // next element in the list
  double remote_time;      // time for the remote node
  double local_time;       // time for the current node
  double offset;         // a node offset to the current node
  double scale_time;       // time for the current noce scaled
  double offset_mean;      // result from mean offset
  double lslr;         // result from LSLR
};
struct NODES    // store data for each NODE
{
    int number_of_node;   // number of the node
    int8_t n;        // current sample
    int8_t reach_max;   // When TRIALS is reached
    struct NODE_INFO info[TRIALS];  // stored data for each TRIALS
    struct NODE_INFO *end;       // GET the oldest TRIAL
    struct NODE_INFO *front;       // GET the newest TRIAL
};

struct RBS_INFO
{
  double realtime;     // time for current node
  double realtime_0;     // first get time for current node used by scale
  struct NODE_INFO *end;  // get oldest TRIAL
  struct NODES nodes[ALL_NODES];  // all nodes
};
```

Figure 5.1: Architecture of the struct and fields

- Second motor controller
- Actuators (used id 4 to enable broadcast)
  - ARM7MC = 0,2

5-3-1 First attempt

At first we tried with a solution where the motor position was first reset to zero. Then the we use getRegisterValueF("CURRENT_POSITION", motorid, value); To receive the position from the motor and setActuatorPWM (speed, motorid); to set the speed of the motor. But this resulted in a lot of complications. First of when trying to reset the motor position the motor would start just moving back and forth very fast, in a way that seemed almost harmful to the motor so we removed that piece of code immediately. It was not of real importance as the simulated motor did not start from zero either. Then there was a problem about using getRegisterValueF(). After having set the PWM value using setActuatorPWM() it was like the setRegisterValueF would just make the motor run wild and stall the program.

5-3-2 Sensor Functions

Then we tried using the sensor functions to see if we had not set up the sensors right in our program to use the tacometer in the motor. From the information in the documentation we made a simple program, see code 5.2, to get sensor information from the tachometer a slice of the most important part is here.

Though the output from the program was not what we had hoped. No sensors were available. So we tried adding the sensors from the example file "readSensors-settings.cfg" from the Locokit documentation. But again it was somewhat of a wall as the Locokit would just not initialize with an error code giving

```
setActuatorPWM(400,motorid);
int numberOfSensors = getNumberOfSensors();
for( i=0; i<numberOfSensors; i++) {
  printf("%s\t\t", getSensorName( i ) );
}
printf("\n");

while(1) {
  for( i=0; i<numberOfSensors; i++ ) {
    printf("%f\t", getSensorValueRawFloat( i ) );
  }
  printf("\n");
  sleep(1);
```

Figure 5.2: Code for getting sensor information from the tachometer

no explanation. Even though we tried modifying the addresses nothing worked. So finally we found a third possibility, getActuatorPosition(motorid);

5-3-3 Third attempt

Finally we used the getActuatorPosition(motorid), together with setActuatorPWM (speed, motorid), to set the speed and get the actuator's position. Here it is easy to get a list of the actuators position when setting the actuator to a constant PWM. So we try to improve the system by creating a full PID controller similar to the controller Rasmus Petersen had created. See code in figure 5.3.

```
setActuatorPWM(500,motorid);
prevError = 0;
errorSum = 0;
while(1) {
  simulated_position = (speed*getTime()/modulation);
  simulated_position = simulated_position - (float)( (int)(
     simulated_position / 360 ) ) * 360;
  double double value = getActuatorPosition(motorid);
  value = value - (float)( (int)( value / 360 ) ) * 360;
  error = (simulated_position-value);
  errorSum = errorSum + error;
  if (error < -180) error = 360+error;
  else if (error > 180) error = error-360;
  double newPWM = (400+(kp * error) + (ki * errorSum) + (kd * prevError))
     ;
  setActuatorPWM(newPWM,motorid);
  prevError = error;
  printf("Current: %f, Simulated: %f, Error: %f, prevError: %f, errorSum:
      %f, PWMvalue: %f\n ", value, simulated_position, error, prevError,
     errorSum, newPWM );
  usleep(10);
}
```

Figure 5.3: Code for PID

The code in black is least needed for a Proportional controller, then for adding an integral part the code in red is added. Then for the Differential control the code in green is added. This is rather standard

for discrete PID control when it is simplified. A time step can also be added if we expect the time ratio between time slices to vary. But for this it can be neglected.

5-3-4 Running into problems

This is when we run into big problems. Even though we do not use any pointers there seem to be some kind of memory leak in the program where variables are set to impossible values and then never set again. An example was where in a code piece with the requirements.

$$If((-200 < (x - prevx))\&(200 > (x - prevx))\&(-1000 < x)\&(1000 > x))newPWM = x; \tag{5.1}$$

This code segment could result in values like -1000345600 being set to newPMW. There were a lot of similar problem. So in general the problem would be what was expected when changing the speed of the motor or getting the position of the motor while also working on setting the speed like we do in our PID controller would often result in the controller being unstable and running of in one direction at max speed and not regulating in to the simulated motor. The problem might be that a PID controller was not the optimal control method for a system like the one we use here or else we just would have needed a lot more time fine-tuning the variables to get the variables to get the control working correctly.

6 Test

6-1 Test RBS

In this section, there all be some screen-shots of the result appeared in the command line. The test is performed on two laptops with wireless. The network used in this test is OdenseKollegienet. The result can be seen in figure 6.1. In the top is y shown which is result of LSLR, it is shown with slope and intercept. Then come the square error and deviation. In PRINT INFO is the information of the received packet. At last is the timestamps and the offsets. The LSLR use 500 observations to compute y in this example.

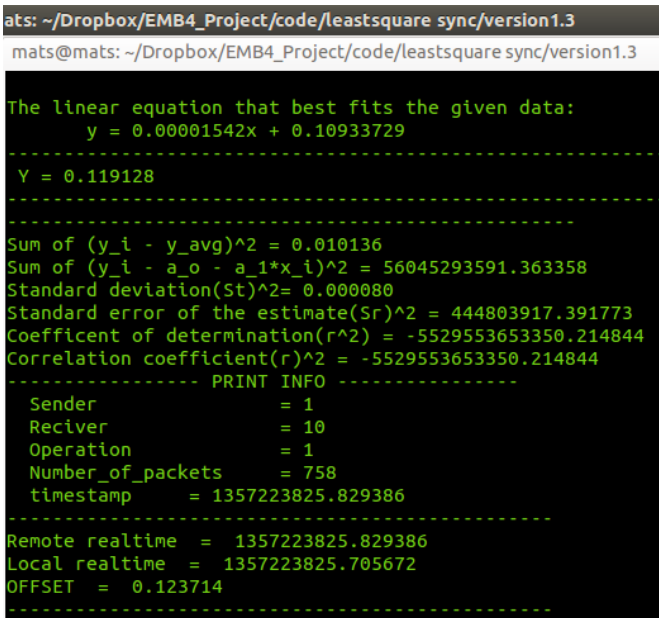

Figure 6.1: Calculation of the least square linear regression

The figure 6.2 shows a list of y values, which can be used to illustrate how these results varies. The variation is roughly 0,002 seconds. It should be mention that the wireless network is used by other users, so the traffic can also vary. Outliers have a heavy negative impact in this regression, so at least 100-200 observations are required to achieve reliable results.

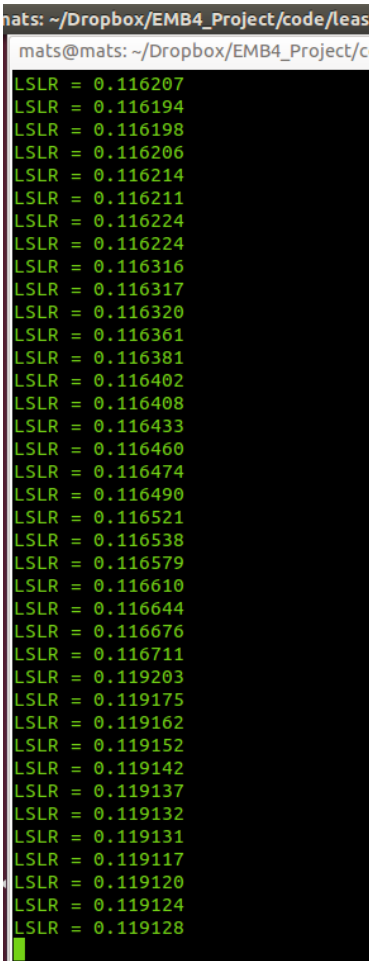

Figure 6.2: List of least square linear regression results. Observations = 500

7 Future Work

Since the results obtained from the RBS method used for the time synchronization of the nodes were reasonably good, some other method could be implemented to compare and test both them, to check if the synchronization could be improved. There are few factors which could be considered for this case:

- Improving the time synchronization by reducing the offset/drift.
- Taking into account time correctness and accuray at each node.
- Considering the time consumption of the program.
- Implementing the time synchronization in a energy-efficient way to reduce the battery consumption.
- Reducing the communication overhead.
- Reducing computational space.

One method has been tried to be used for this purpose: Self-Correcting Time Synchronization. Nevertheless, and due the fact that the method was not completed and running before the deadline, it has been explained theoreticaly, in a way that it shows a total understanding of how it works and how it should be implemented. Therefore, it was just a matter of time having it ready to performance some tests.

7-1 Self-Correcting Time Synchronization

The Self-Correcting Time Synchronization (SCTS) method has been developed by using reference boadcast focused for wireless networks, and it mostly fulfil all bullets mentioned previously in this section. It basically converts this time synchronization problem into an online self-adjusting optimizing problem, where each node is in charge of synchornize itself in function of the sender timestamp. Unlike the RBS method, for the SCTS, both offset and drift compensation are made simultaneously at each node.

7-1-1 Messaging

In order to reduce or to control the communication overhead, and considering possible bandwidth limititations in the network, the message exchanges has to be rather low. This brings the choise of using unidirectional broadcasting instead of pair-wise, where a central node broadcasts its own and reference timestamp to other nodes for them to synchronize autonomously. It can be easily checked how for the first synchronization method, the number of exchanged messages for synchornizing in a group of n nodes will be $(n\text{-}1)$, inspite of the $2(n\text{-}1)$ used for any pair-wise.

7-1-2 Offset & Drift

Both offset and drift are inherited from the characteristics of each node, what means each one of them has its own local clock. Therefore, for this method the node clocks are assumed not to be accurate, with

different drifts (a_n) with constant values. It means they describe their same respective function during the time, what can be seen of figure 7.1 for $c_i(t)$ (central node), and $c_k(t)$ (any other node). Note how the offset $c_k^o(t)$ and the drift $c_k^d(t)$ will always describe the same function over the time, independent of the chosen synchronization point, and the longer the period between these synchronization points, the larger the error is.

The drift $c_k^d(t)$ is calculated using the drift of each one of the local clock functions from each node, assuming their respective drifts are accurately estimated, as following $a_{ik}(t) = a_i(t) - a_k(t)$, which are obtained from the angle difference with the estimated *perfect clock*. Note that the frequency of $c_k^d(t)$ is independent of the local clock $c_k^d(t)$, but which tries to approximate to the one from the reference clock $c_i(t)$.

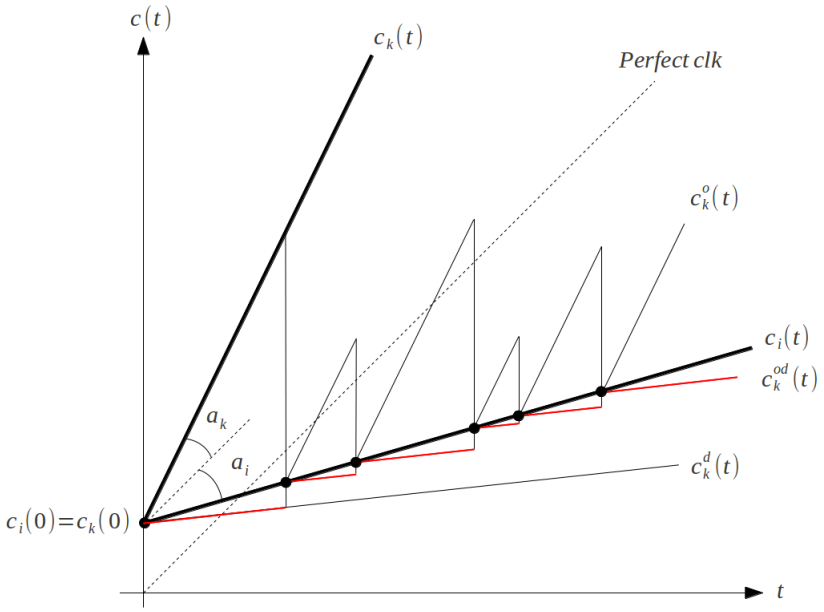

Figure 7.1: Offset and drift compensation.

The result of the combination of both compensations together (offset and drift) is the synchronized clock $c_k^{od}(t)$.

7-1-3 Implementation

The main goal of this method is to use a sequence of successive broadcast reference packets to drive the synchronized clocks on the receiver nodes to gradually approach and eventually be locked to the reference clock on the beacon node. It means that all the nodes in a broadcast domain will be eventually synchronized.

The synchronization of the nodes is made by using a phase locked loop (PLL), and its main elements are shown on figure 7.2. The synchronization will follow the next steps:

- The beacon node will broadcast its local clock timestamps or reference clock to the receiver nodes.
- The error between the reference clock and the local clock of the receiver enters the loop filter to eliminate possible noise.
- The control variable manages the frequency of the VCO.
- The pulse counter counts the obtained pulses to synchronize the clock.

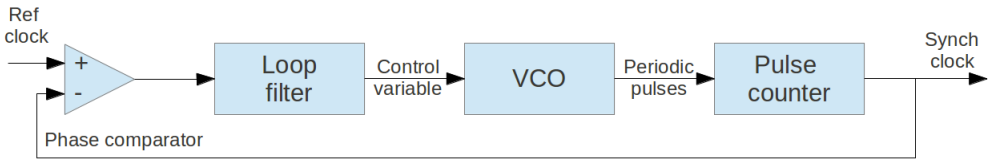

Figure 7.2: Components of the PLL for clock synchronization.

On the paper, the loop filter has been implemented using a PI filter, in which the proportional part is only in function of the proportional constant K_p, and the integrative part is in function of the constant parameter K_i, the synchronization period T, and the zeros and poles, as shown on figure 7.3. Due there is no need to introduce extra hardware to the system, and so to the LocoKit, the VCO has been substituted by a constant frequency parameter K_0 which is unique at each local node due the frequency of its crystal oscillator. An equivalent block diagram is obtained from this statements and can be seen of figure 7.3 (b).

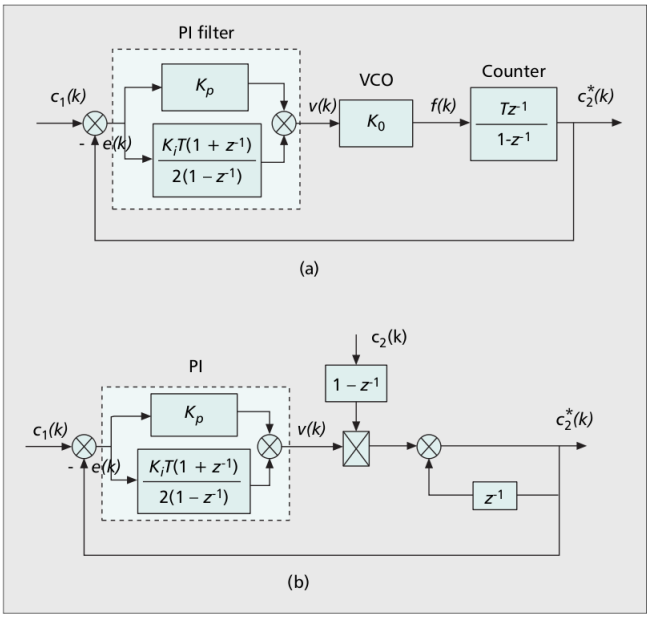

Figure 7.3: The digital phase locked loop: a) the block diagram of a digital PLL; b) an equivalent of the digital PLL without VCO.

7-2 Power management

When dealing with wireless sensor networks which are battery powered, power savings are an essential factor. One another part of the future work will be to implement power management, to achieve that nodes are active in a minimum of time to produce all work, which are needed and then return to a suspended state. Linux kernel in these nodes support power management, so it is a question to get it integrated. Some of the benefits of time synchronization is that all nodes can be active at the same time which will optimize the power savings. All these communication process are a huge power consumer, and this has to be reduced.

8 Conclusion

According to the synchronization implementation, we think that this RBS way is a bit more effective and precise than the others. We also implement least square linear regression which we think is a better solution than the mean of the offsets and it is also implement reasonably way. But still the least square linear regression is a heavy computing task, if we have in mind that the device is power limited and should be energy effectively. It depends on the application, because in some cases the mean of offsets will be an acceptable solution. Mention in future work, power management is required part in power limited systems and it needed to be part of the system to have an energy effectively system. This project has learn us, that it is not possible to obtain extreme precise clocks over wireless, there will always be a small variation.

The same time problem was apparent with the motor control. With extra time we probably would have been able to make the motor control work and implement it with the SCTS so it would be possible to see the whole system work together but running into a lot of problems with the Loco kit motors was not what we had hoped for. But it was a great experience to work with the kit and it defiantly had a lot of possibilities. But for the time being there is also a lot of complications with getting it working probably.

If this method(Self-Correcting Time Synchronization) would have been implemented, different test could have taken place in order to compare it with the RBS implemented and tested before. One of the most important advantages this method provides is that since it does not need from any number of previous storaged synchronization points, the error is more rapidly reduced, and the real-time synchronization is achieved right the way. Besides that, and since the synchronization is autonomously, the nodes only take into account the timestamp of the local node and its own local clock, so they do not consider the other nodes at all. This is a relevant advantage regarding communication overhead and bandwidth limitations on this kind of networks. Once again, and due time issues, this method was not completed, so there was not a possibility of testing it.