Lego Mindstorm Project: Lego Segway
FRTN01 - Real time systems

Simon Wallström, dat11swa@student.lu.se
Adam Dalentoft, dat11ada@student.lu.se
Jonathan Karlsson, ada09jka@student.lu.se

Supervisor: Victor Millnert

December 17, 2014

Contents

1 Introduction **2**

2 Program structure **3**

3 Control design **4**

4 User information **6**

5 Results **7**

5.1 Parameters and values 7

5.2 Hardware 8

5.3 PID Controller 9

5.4 Feedback Controller 10

6 Conclusion **10**

1 Introduction

The goal of this project is to build and control a two-wheeled robot - a Lego Segway - using NXT Lego Mindstorm 2.0. A segway is an unstable system that constantly needs to be regulated to work. To regulate this system several parameters is required. It is important to know the mass and specifications of the hardware that is used and the physics and movements in the system. Basically how it works is that the the robot tilts and the angle is measured. Thereafter the angle is processed by the controller to calculate the required change in power and direction of the motors. It is possible to control the segway in a few different ways. One way is to use pole placing and feedback control. Even though this is a good way to regulate the system it is not the first choice. Instead, the structure that will be described in this report uses two PID controllers, one for the wheel position and one for the angle of the robot. Later, the structure is changed to a state feedback controller due to lack of time, more about this later.

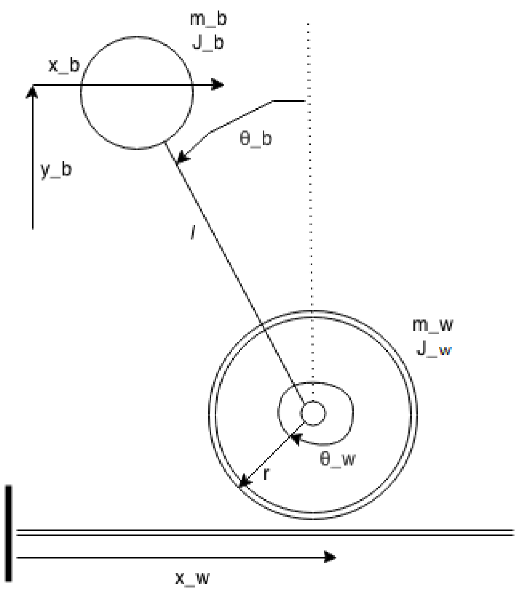

Figure 1: Model over the segway movements, all variables and their names are set in the picture. Values on the variables is given in the results.

Before the implementation can begin it is a good idea to set up a model of the process. The process is described in the picture, including the different variables that is needed to calculate the movements in the system. These variables are then used to calculate the state-space form for the system, describing the equation L = T - V. The state-space form of the system can thereafter be used in matlab and simulink to describe a control system for the segway. Simulink can also be used to make a simulation of the calculated system to determine parameters for the PID-controllers.

To implement the controller a Java version for Lego NXT is used called LeJOS. There are several alternatives in other languages but this is the only widely known Java VM. LeJOS provides classes for the most common hardware to the Lego NXT[1].

To measure the angles a gyroscope and an accelerometer are used. The gyroscope measure the angle velocity and the integrated value gives the angle of the tilted segway. However the gyroscopes raw value drifts and therefore the accelerometer is required to compensate for this. Rest of the hardware is standard nxt motors and base unit.

2 Program structure

The program uses threads to realize a real time controlled system. The LeJOS VM provides support for real time threads and synchronization, the same as standard Java. There is a variety of classes building up the program, these are described in the diagram below. To clarify the meaning of the different classes there is also a brief description of them.

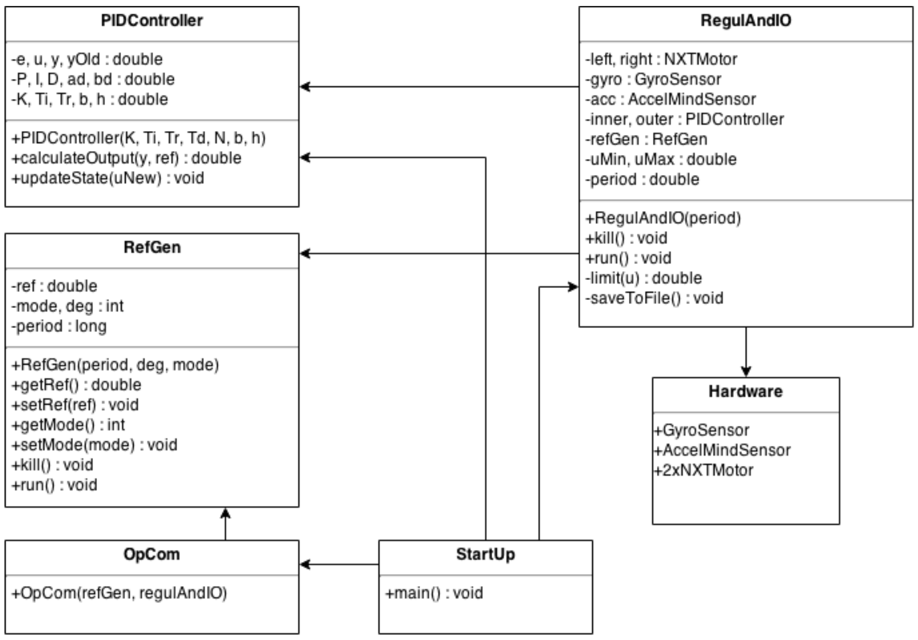

Figure 2: UML diagram of the classes for the PID controller.

- StartUp: main class, starts all the other classes and threads.
- OpCom: Operator communication class, handles user input through passive listeners on the buttons on the base unit. OpCom also receives parameter updates from a command line interface. Some of the commands are for control variable change and program exit.
- RefGen: Reference generator. Generate reference values for input to the robot.

- RegulAndIO: Control class and hardware communicator. Uses the PID-Controller class together with the hardware input and output to run and balance the segway.
- PIDController: Controller class. Calculate the P, I and D values using a standard control equation from lecture 8[2].
- Hardware: All the hardware connected to the software together with their in and output methods.
- FeedbackController: Controller class for the feedback controller. Not included in the original design and replaces RegulAndIO and PIDController. Otherwise it is connected in the same way as those classes.

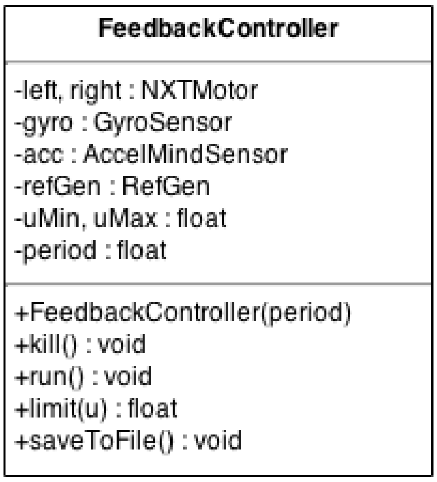

Figure 3: UML diagram of the Feedback class that is used instead of the RegulAndIO class.

3 Control design

As mentioned in the introduction the controller will contain two PID-controllers. They are connected in a cascade to regulate the two angles: tilt angle and wheel angle (rotation of the wheel). The inner loop controls the tilt angle by moving the wheels and the outer loop controls the wheel position in case of a reference change and to make sure the robot doesn't drift when standing still.

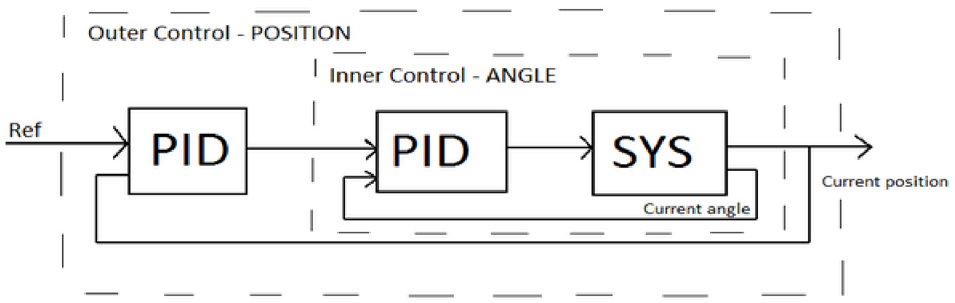

Figure 4: Figure over the PID controllers. The inner loop controls the angle of the segway and the outer the position of the wheels.

The main reason for this choice of control structure is the rather easy way to implement it. As mentioned in the introduction there are other ways to control the system, but this method has been used earlier in the course which made it the first choice and more natural to implement.

The figure describes how the control structure works. First the reference value is sent in. It can either be 0, to stand still, or a value decided by the reference generator. The mode is decided by the user (further description in the user information section). This value describes the desired position of the segway. Thereafter the value, together with the feedback values from the previous calculations, is processed by the two PID-controllers. When the segway should stand still only the inner loop is required.

An optional solution is to use state feedback. Here the different measurements (angle, position, angular velocity and wheel movement) of the system is fed back and thereby the different states of the system is controlled based on previous location and value. This version got a structure like the second figure in this section. The phi value controls the angle of the robot, phi dot the angular velocity, theta the wheel movement and theta dot the wheel velocity. This method is used in the final solution.

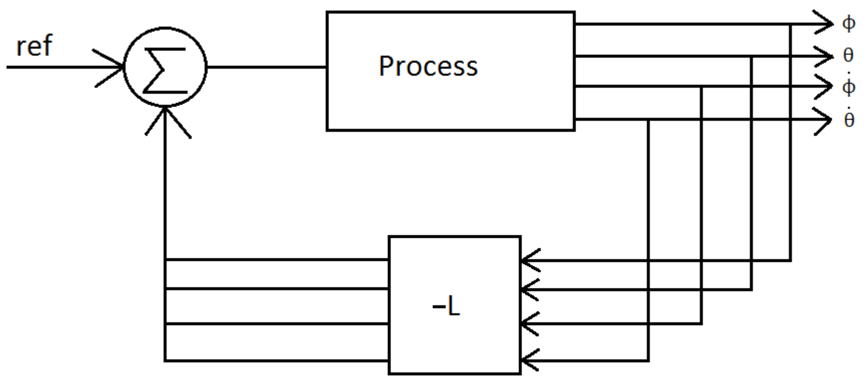

Figure 5: Figure over the control scheme of the state feedback version of the controller.

4 User information

The segway do not have a user interface except the small screen and the buttons on the front of the base unit. When the segway is in motion these become a rather impractical way of controlling the system. Therefore the system is started before it is put down. A press on the dark grey button, the Escape button, will quit the program. The screen give out small messages to inform the user how to handle the robot and what is happening in the system.

The program should be pre-installed on the machine. To run it is done in a few simple steps.

1. First start the segway using the large yellow button, the Enter button.
2. On the screen a menu will be displayed. Choose the default program on the screen.
3. Now the program starts. A message will be displayed, "Calibrating...". It is important to hold the robot stationary, otherwise the calibration of the hardware may be disturbed.
4. After the calibration, the message "Calibrated, hold robot and press Enter to balance" is shown. Now, put down the robot on the ground and press the Enter key.
5. After the massage "Balancing" is displayed the segway will start to balance, hopefully.
6. To terminate the program push the Escape key.

5 Results

5.1 Parameters and values

Model values

$J_w = 1.6 * 10^-5$ Wheel inertia
$R_w = 0.08$ Wheel radius
$L = 0.0950$ Length from wheel to body mass
$m_b = 0.5120$ Body mass
$m_w = 0.031$ Combined mass of both wheels
$J_b = 0.0019$ Body inertia
$b = 0.062$ Damping factor
$g = 9.81$ Gravity

L values

$L_1 = -10$
$L_2 = 0$
$L_3 = -0.355$
$L_4 = -0.225$

PID values (experimental, do not work)

$K = 9$
$Ti = 2$
$Tr = 1$
$Td = 0.1$
$N = 10$

Filter equations

Complementary: $GyroAngle * 0.98 + AccelerometerAngle * 0.02$
Low pass: $G_{lp} = \frac{1}{s+1}$
High pass: $G_{hp} = \frac{s}{s+10}$

5.2 Hardware

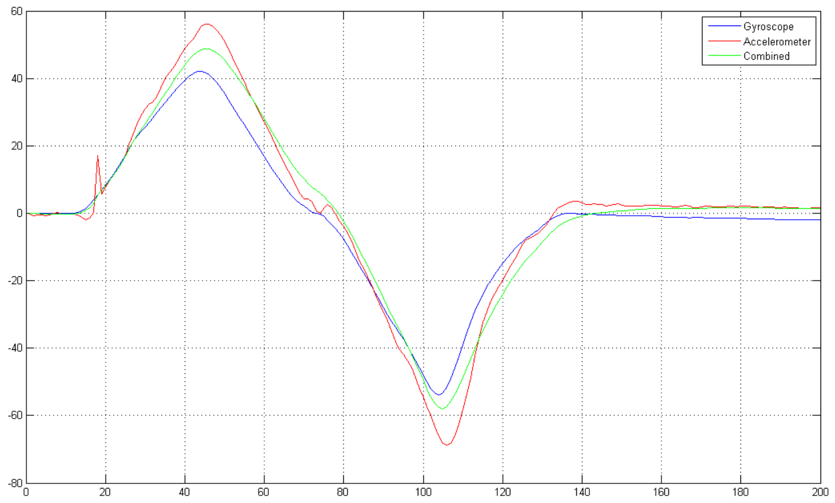

Figure 6: The figure describes the relation between the gyroscope and the accelerometer, together with the complementary filter.

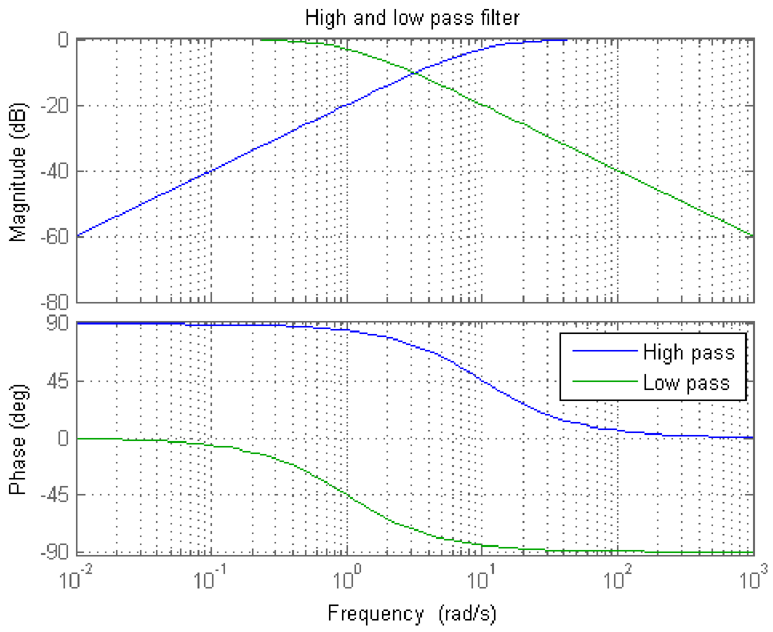

Figure 7: High and low pass filter used to filter signals from gyroscope respectively accelerometer.

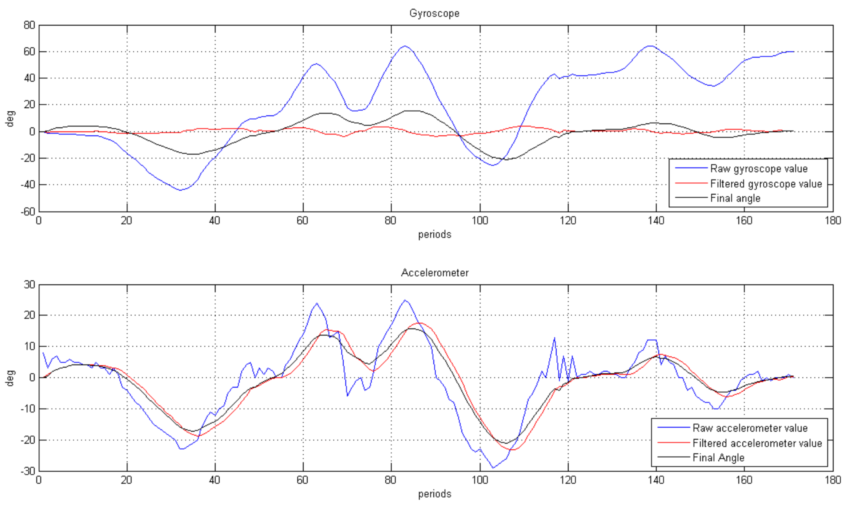

Figure 8: High and low pass filter used to filter signals from gyroscope respectively accelerometer.

5.3 PID Controller

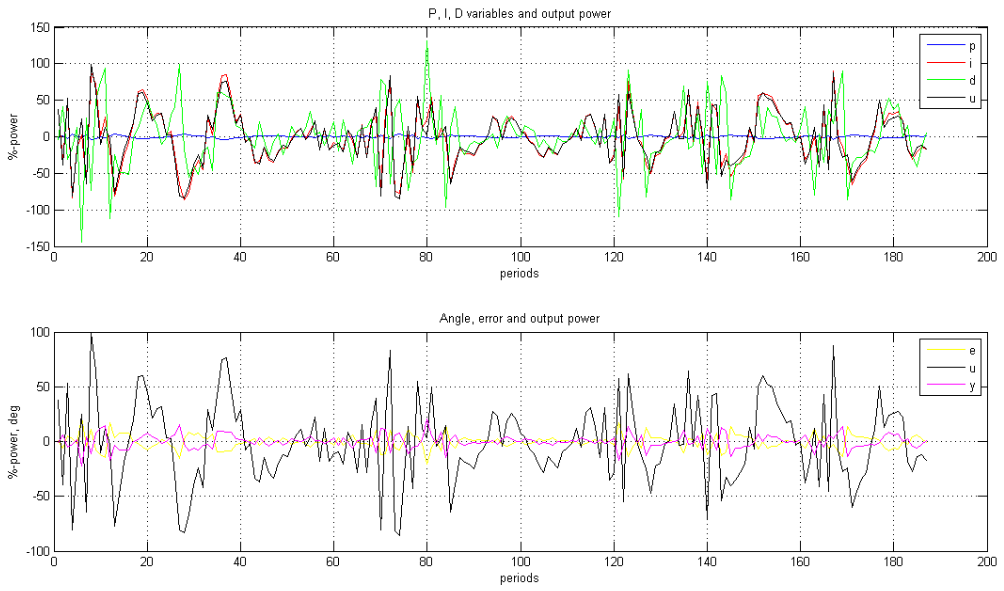

Figure 9: Relation between the different parameters in the PID controller.

5.4 Feedback Controller

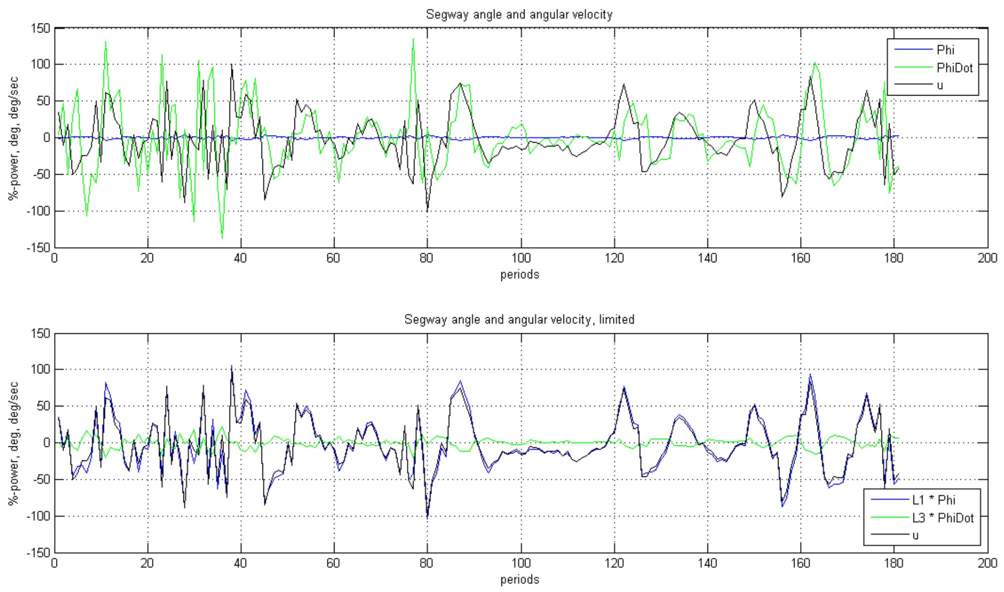

Figure 10: Relation between the different parameters in the feedback controller for the angle of the segway.

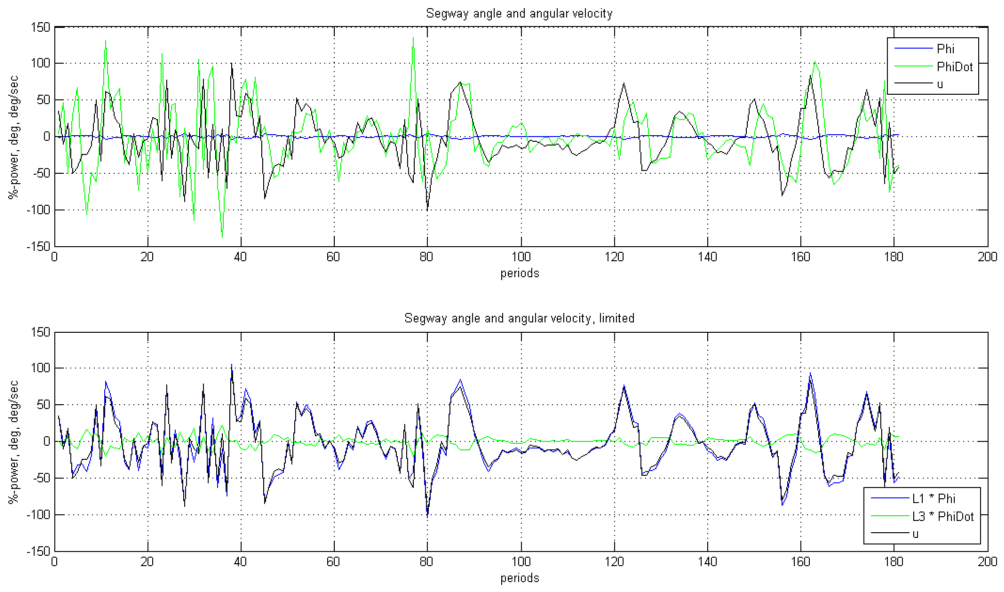

Figure 11: THETA ISTÄLLET!

6 Conclusion

The project in itself is a good example of real-time-programming and control. To complete the project it is required to have knowledge in both of the areas. Even though implementing a segway is a common control task (both as a Lego Mindstorms project and others) is it not an easy task. There are numerous of factors that can go wrong.

As for all realizations of theory there are error sources, this is no exception. To start with, the hardware is not perfect. As previously mentioned the gyroscope drifts, that is because of temperature shifts, among other things. This was originally solved by using the gyroscope and an accelerometer in a complementary filter to smooth out the measurements and thereby eliminate the drift. The final solution is to filter the gyroscope value in a high pass filter to get rid of the drift over time and add a low pass filtered accelerometer value to get rid of the flickering in sudden changes. Even though the gyroscope problem is solved the accelerometer introduce another problem by adding a couple of milliseconds to the control loop. Thereby the period has to be increased which makes the whole system slower and harder to control. The rest of the hardware can also give errors, such as the motors being unsynchronized, cable problems and computing errors in the base unit[3].

The software implementation can also give some errors, especially the parameters of the controllers. Tuning the parameters is extremely time consuming and even the smallest change can give a large difference.

Because of the many error sources the state feedback solution may give a more stable control. The new solution is the one previously discussed, a feedback loop where the different states of the segway is used to control the movements. This solution is easier to tune and therefore less time has to be spent, especially good when time is of the essence. In terms of control, there is no considerable difference. To compare the plots from the two is a bit hard because of the different parameters used when controlling. However, because of the differences this gives a good indication on how systems can be controlled in many different ways with the same results, it all comes down to preferences.

Because of the many problems and the change of controller the result of the project was not the expected one. Many changes had to be made along the way to try fix the errors constantly occurring. The implementation had different stages where different combinations of threads and monitors were used, mostly because of optimization to fit the real time requirements. The most time consuming part of the project was tuning the PID parameters, this part felt very tiresome and much work made little progress. Because of this the decision to change the control design to a state feedback was considered, because the segway had problems stabilising with the PID controller. This new method was rather new and unexplored because all the preparations and simulations were made matching the first solution. Things that could have been made different is things like research on different controllers before the implementation and take more basic things in consideration like hardware problems. The project being successful was partly because of an implementation that made it possible to update the control parameters in real time without having to recompile for each change. If this would have been implemented earlier it would have reduced the time spent on tuning.

References

[1] LEJOS NXT, Java for LEGO Mindstorms
`http://www.lejos.org/nxj.php`
(Hämtad 14/12-14)

[2] Lecture 8, Real-Time Systems LTH, 23/9-14
`http://www.control.lth.se/media/Education/EngineeringProgram/FRTN01/2014/L8_14.pdf`
(Hämtad 14/12-14)

[3] The Balance Filter, Shane Colton, 25/6-07
`https://b94be14129454da9cf7f056f5f8b89a9b17da0be.googledrive.com/host/0B0ZbiLZrqVa6Y2d3UjFVWDhNZms/filter.pdf`
(Hämtad 10/12-14)