Self-Balancing Lego Robot

Mattias Josefsson*, Jonathan Persson†, Alexander Skafte‡ and Jakob Warlin§
Supervisor: Tommi Nylander
Lund University
*tpi13mjo@student.lu.se, †ast13jpe@student.lu.se, ‡tfy13ask@student.lu.se,
§tfy13jwa@student.lu.se, https://gitlab.control.lth.se/regler/FRTN40/2017/group-B

***Abstract*—The aim of this project was to design and implement different control strategies for balancing a robot on two wheels, and to investigate how measurement noise affects the performance. The problem can be formulated as balancing an inverted pendulum, which is a standard nonlinear control problem. PID and linear-quadratic control strategies were implemented and compared to each other. In order to measure the angle of the robot a gyroscope and an accelerometer were used, and measurement noise was filtered using two different methods, which we compared to each other: Kalman filtering and complementary filtering. The control algorithms were implemented first in Simulink and then in leJOS on a LEGO Mindstorms EV3 microprocessor. Our findings suggest that an LQR manages to balance the robot better than a PID controller does. As for filtering, our findings suggest that a Kalman filter is the best filter to use in order to reduce the drift from the gyroscope and the noise from the accelerometer.**

I. Introduction

This project aimed at balancing a two-wheeled LEGO structure, shown in figure 1, similar to a Segway by keeping it in the upright position. The control problem is that of an inverted pendulum. Different control strategies for keeping the robot upright and steering it have been implemented and tested. Two common control methods have been compared: optimal control through a *linear-quadratic regulator* (LQR), and *proportional-integral-derivative* (PID) control.

II. Equipment and materials

The robot was built using LEGO Mindstorms. The main components of the robot are two wheels, a LEGO EV3 processor, two NXT motors, a gyroscope and an accelerometer. Other LEGO pieces were used to construct the frame. The regulator for the robot was implemented in Java using the Eclipse IDE. The program was compiled on a computer connected to the robot with a USB-cable. The robot structure is partly based on blueprints found online [4].

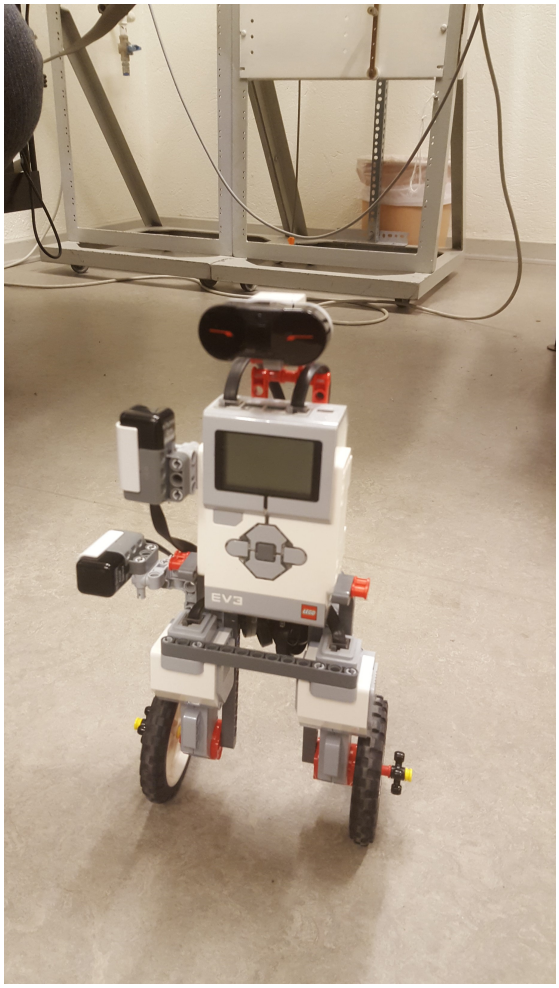

Fig. 1. Lego model

III. Theory

In this section theoretical backgrounds for the different control theories are described.

A. Complementary filter

When dealing with multiple sensors with various flaws, a complementary filter can be used to minimize these flaws. The idea of a complementary filter is to combine filtered signals from different sensors in such a way that the desirable properties of each sensor are utilized. An example is to combine an accelerometer and a gyroscope to measure an angle: a gyroscope typically is inaccurate for low frequencies, which causes measurement drift, and an accelerometer typically has undesirable high frequency components, which causes measurement noise. By numerically integrating the gyroscope's signal and passing it through a high-pass filter and combining this signal with a low-pass filtered accelerometer signal, an estimate of the angle can be retrieved. See figure 2. This can easily be implemented using the update formula in equation 1.

$$angle = a \cdot (angle + gyro \cdot dt) + (1 - a) \cdot accel, \quad (1)$$

where

$$a = \frac{\tau}{\tau + dt},$$

where τ is the desired time constant and $dt = \frac{1}{fs}$ where fs is the sampling frequency.

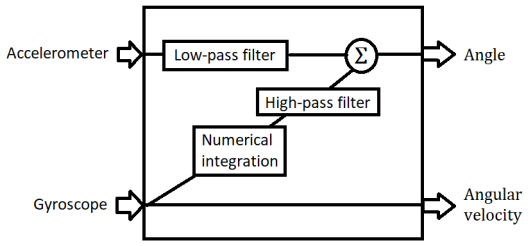

Fig. 2. Complementary filter

B. Kalman filter

A Kalman filter is an algorithm that can be used in order to estimate unknown variables and reduce noise from measurements. For a discrete-time state-space model on the form

$$x_{k+1} = Ax_k + Bu_k + v_k \quad (2)$$
$$y_k = Cx_k + e_k, \quad (3)$$

where $E\{v_k v_j^T\} = Q_v \delta_{kj}$, $E\{e_k e_j^T\} = Q_e \delta_{kj}$, and $E\{v_k e_j^T\} = Q_{ve} \delta_{kj}$, the Kalman filter is constructed according to equations 4-8: [2]

$$\hat{y}_k = C\hat{x}_k \quad (4)$$
$$R_k = Q_e + CP_kC^T \quad (5)$$
$$P_{k+1} = AP_kA^T + Q_v - K_kR_kK_k^T \quad (6)$$
$$K_k = (AP_kC^T + Q_{ve})(Q_e + CP_kC^T)^{-1} \quad (7)$$
$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + K_k(y_k - \hat{y}_k). \quad (8)$$

In the case of estimating the angle with measurements from an accelerometer and a gyroscope, a simpler Kalman filter can be implemented according to the following equations: [3]

$$\begin{aligned}
\hat{\mathbf{x}}_{k|k-1} &= \mathbf{F}\hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}\dot{\theta}_k \\
\mathbf{P}_{k|k-1} &= \mathbf{F}\mathbf{P}_{k|k-1}\mathbf{F}^T + \mathbf{Q}_k \\
\hat{\mathbf{y}}_k &= \mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1} \\
\mathbf{S}_k &= \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R} \\
\mathbf{K}_t &= \mathbf{P}_{k|k-1}\mathbf{H}^T\mathbf{S}_k^{-1} \\
\hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\hat{\mathbf{y}}_k \\
\mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_{k|k-1}, \quad (9)
\end{aligned}$$

with

$$\hat{\mathbf{x}}_k = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} 1 & -h \\ 0 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} h \\ 0 \end{bmatrix}$$
$$\mathbf{Q}_k = \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad \mathbf{R} = var(e_k), \quad (10)$$

where θ is the angle, $\dot{\theta}_b$ the bias of the measurements from the gyro, h the sample time, Q is process noise, and R measurement noise. The noises are the parameters that can be tuned. The initial P-matrix can be set to a 2x2 matrix with zeros.

C. Linear-quadratic regulator

The LQR was used as the primary regulator. It is concerned with operating a dynamic system at minimum cost and can be described by a set of equations:

$$u(t) = -L\hat{x}(t) \quad (11)$$
$$\dot{\hat{x}}(t) = A\hat{x}(t) + Bu(t) + K(y(t) - C\hat{x}(t)) \quad (12)$$
$$\begin{aligned} 0 = {} & AP + PA^T + NR_1N^T \\ & - (PC^T + NR_{12})R_2^{-1}(PC^T + NR_{12})^T \end{aligned} \quad (13)$$
$$K = (PC^T + NR_{12})R_2^{-1} \quad (14)$$
$$L = Q_2^{-1}B^TS \quad (15)$$
$$0 = A^TS + SA + M^TQ_1M - SBQ_2^{-1}B^TS \quad (16)$$

The equations eventually yield a state feedback controller.

D. PID control

A PID controller finds the difference between the desired reference signal and the actual system output signal and thereafter calculates a control signal. Denoting the error as $e(t)$ the PID-controller on standard form can be written as:

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(t)\,\mathrm{d}t + T_d\frac{\mathrm{d}e(t)}{\mathrm{d}t}\right),$$

or, as a transfer function in the Laplace-domain (s-domain)

$$G(s) = K\left(1 + \frac{1}{T_i s} + T_d s\right),$$

where K, T_i and T_d are tuning parameters for the controller. Since the controller was implemented digitally a discrete-time representation was needed. Using the backward Euler method with sampling time T_s the following transfer function was acquired:

$$H(z) = K\left(1 + \frac{T_s}{T_i}\frac{1}{1 - z^{-1}} + \frac{T_d}{T_s}(1 - z^{-1})\right)$$

Letting $K_1 = K$, $K_2 = T_s/T_i$ and $K_3 = T_d/T_s$ the equation can be rearranged into:

$$\begin{aligned} H(z) &= \frac{K_1(1 - z^{-1}) + K_2 + K_3(1 - z^{-1})^2}{1 - z^{-1}} \\ &= \frac{(K_1 + K_2 + K_3) + (-K_1 - 2K_3)z^{-1} + K_3z^{-2}}{1 - z^{-1}}. \end{aligned}$$

Let $K_a = K_1+K_2+K_3$, $K_b = -K_1-2K_3$ and $K_c = K_3$, yielding:

$$u[k] = u[k-1] + K_a e[k] + K_b e[k-1] + K_c e[k-2],$$

which is a causal assignment that can be implemented in computer code.

IV. Modelling

The dynamics of a Segway, which the robot was based on, can be described as follows:

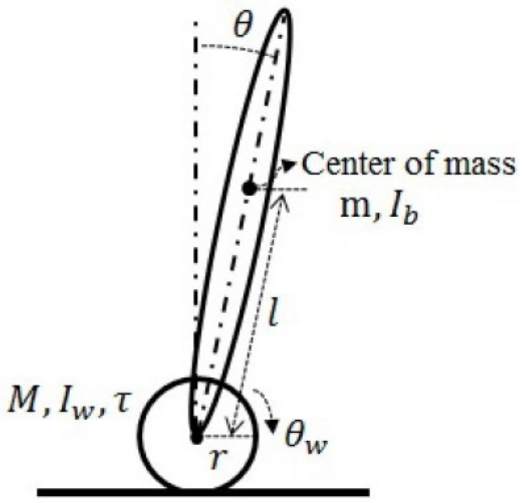

Fig. 3. Segway dynamics [1]

$$\begin{cases} m_{11}\ddot{\theta}_w + m_{12}\ddot{\theta}\cos\theta = \tau + m_{12}\dot{\theta}^2\sin\theta \\ m_{12}\ddot{\theta}_w\cos\theta + m_{22}\ddot{\theta} = -\tau + G_b\sin\theta \end{cases}, \tag{17}$$

with:

$$\begin{aligned} m_{11} &= (m+M)r^2 + I_w \\ m_{12} &= mlr \\ m_{22} &= ml^2 + I_b \\ G_b &= mgl, \end{aligned} \tag{18}$$

where M and m are the masses of the wheel and the body, r the radius of the wheel, l the length from the wheel to the centre of gravity, θ_w the rotational angle of the wheel, θ the angle of the body, I_w and I_b the moment of inertia for the wheel and the body respectively, and τ the applied torque on the wheel, as seen in figure 3. [1]

A. First model

Introducing the states $x_1 = \theta$ and $x_2 = \dot{\theta}$ yields:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = f(x_1, x_2, \tau) \end{cases}, \tag{19}$$

where:

$$f(x_1,x_2,\tau) = \frac{m_{11}G_b\sin(x_1) - m_{12}^2\cos(x_1)\sin(x_1)x_2^2 - m_{11}\tau}{m_{11}m_{22} - m_{12}^2\cos^2(x_1)}. \tag{20}$$

The system in equation 19 can be linearised around the point $(\theta,\dot{\theta}) = (0,0)$, giving the system:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \frac{m_{11}G_b}{m_{11}m_{22}-m_{12}^2}x_1 - \frac{m_{11}}{m_{11}m_{22}-m_{12}^2}\tau \end{cases}. \tag{21}$$

Finally by introducing $N = \frac{m_{11}}{m_{11}m_{22}-m_{12}^2}$, the system can be written in matrix form as

$$\begin{cases} \dot{x} = Ax + B\tau \\ y = Cx \end{cases}, \tag{22}$$

with:

$$A = \begin{bmatrix} 0 & 1 \\ NG_b & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ -N \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 1 \end{bmatrix}. \tag{23}$$

This model would later turn out to be insufficient and another model with more states was tested.

B. Second model

From equation 17, the angular acceleration of the wheels can be written as

$$\ddot{\theta}_w = \frac{(m_{12}\cos\theta + m_{22})\tau + m_{22}m_{12}\dot{\theta}^2\sin\theta - m_{12}G_b\sin\theta\cos\theta}{m_{11}m_{22} - m_{12}^2\cos^2\theta}. \tag{24}$$

Linearising this equation around $(\theta,\dot{\theta}) = (0,0)$ gives

$$\ddot{\theta}_w = \frac{-m_{12}G_b}{m_{11}m_{22}-m_{12}^2}\theta + \frac{m_{12}+m_{22}}{m_{11}m_{22}-m_{12}^2}\tau. \tag{25}$$

Using equation 25, the system from equation 22 can be extended to include the wheel's angle and angular velocity as states. with the state vector $\mathbf{x} = \begin{bmatrix}\theta & \theta_w & \dot{\theta} & \dot{\theta}_w\end{bmatrix}^T$ and by introducing $K = m_{11}m_{22} - m_{12}^2$, the system become:

$$\begin{aligned} \dot{x} &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ NG_b & 0 & 0 & 0 \\ \frac{-m_{12}G_b}{K} & 0 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ -N \\ \frac{m_{12}+m_{22}}{K} \end{bmatrix}\tau \\ y &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x. \end{aligned} \tag{26}$$

V. Control

In order to implement this on the real process, the system has to be discretized. This can be done with zero-order hold sampling. For the first model, the system becomes

$$\begin{aligned} x(t_k+h) &= \Phi_h x(t_k) + \Gamma_h u(t_k) \\ y(t_k) &= Cx(t_k), \end{aligned} \tag{27}$$

where

$$\Phi_h = \begin{bmatrix} \cosh(\sqrt{NG_b}h) & \frac{\sinh(\sqrt{NG_b}h)}{\sqrt{NG_b}} \\ \sqrt{NG_b}\sinh(\sqrt{NG_b}h) & \cosh(\sqrt{NG_b}h) \end{bmatrix}, \quad (28)$$

$$\Gamma_h = \begin{bmatrix} -\frac{\cosh(\sqrt{NG_b}h)-1}{G_b} & -\frac{\sqrt{N}\sinh(\sqrt{NG_b}h)}{\sqrt{G_b}} \end{bmatrix}^T, \quad (29)$$

and $h = 0.02$ is the sample time. For the second model, the matlab function *c2d* was used.

The linear systems can now be controlled using a LQR (described above) or a PID controller.

VI. Implementation

A. Simulation

In order to test the control strategies and the different filters before moving on to the real process, Simulink was used. Two different models were created, one with a complimentary filter, figure 5, and one with a Kalman filter, figure 5. Both used LQR to control the system.

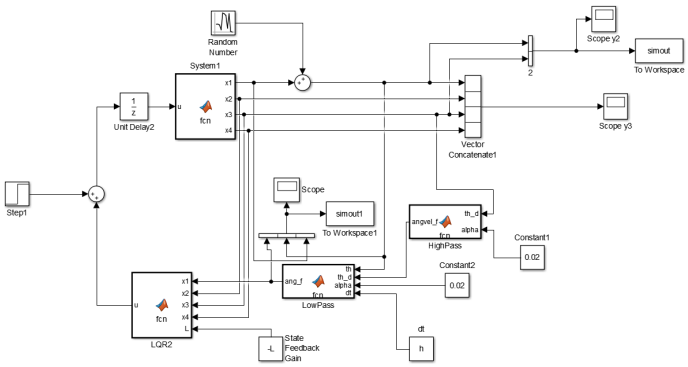

Fig. 4. Simulink model of the linearised and discrete system (with 4 states) with a LQ controller and a complementary filter.

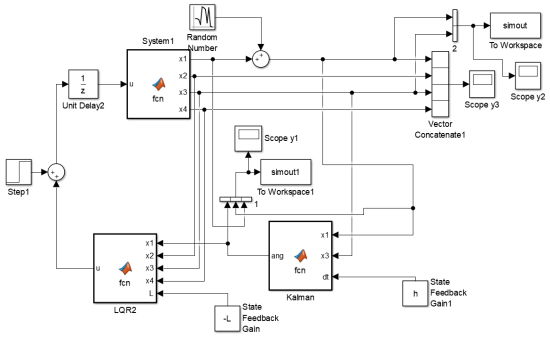

Fig. 5. Simulink model of the linearised and discrete system (with 4 states) with a LQ controller and a Kalman filter to estimate the angle.

In figure 4, the angle is filtered using a complementary filter with both the angle and the angular velocity as inputs. The filter is based on figure 2, where both the integration and the low-pass filter is included in the *LowPass* block. The system is controlled using a LQR with parameters found from the matlab function *lqr*. There is also a step disturbance on the control signal and noise on the measured angle to represent a bad sensor.

The model in figure 5 is very similar, but instead of a complementary filter, there is a Kalman filter described by equation 9 to estimate the angle.

B. Java

The program for the robot was written in Java and implemented with the operating system leJOS, which is a Java based operating system. leJOS VM supports most of the functionality from the standard library such as real-time threads, synchronization mechanisms, and it also includes some libraries for the LEGO Mindstorms hardware. In figure 6 the structure for the classes used for controlling the robot and communicating with the computer are visualized. A conceptual description of the blocks (Java classes) are described below:

- Main: Main method, starts all the required threads.
- ControlRobot: Depending on chosen control strategy, this controller regulates the dynamics of the robot.
- Wifi: Used for communication between the robot and a computer on the same LAN.
- LQR and PID classes: one class for each control strategy. Stores the variables needed for the control strategy.

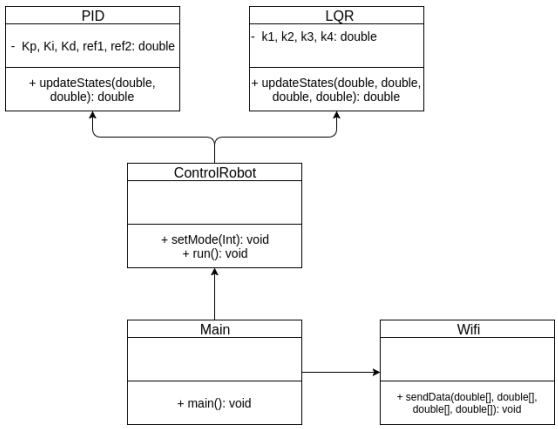

Fig. 6. UML diagram of the program structure

VII. Method

An LQR controller with feedback was tried with the states:

$$x = (\theta,\, \dot{\theta}), \quad (30)$$

where θ is the angle of the robot body and $\dot{\theta}$ is the angular velocity of the body. However, after trial and error the robot still did not manage to balance, so another model was made with 4 states:

$$x = (\theta,\, \theta_w,\, \dot{\theta},\, \dot{\theta}_w), \quad (31)$$

where the added states θ_w and $\dot{\theta}_w$ are the wheel angle and the angular velocity of the wheel. Following a trial and error approach by simulating the system in Simulink and then tuning the parameters on the real process, new model parameters were found that managed to balance the robot to a sufficient

degree. With the robot now successfully balancing, using only a gyroscope to measure the angular velocity and the angle by numerical integration, and the wheels to measure the current wheel angle and wheel angular velocity, an accelerometer was added to calculate the angle using sensor fusion. A complementary filter was implemented that mostly trusted the value from the gyroscope due to the accelerometer value being corrupted by noise. Secondly, to improve the measurements from the gyroscope and accelerometer for the angle, a Kalman filter was implemented according to equation 9. Simulations were done in Simulink with the different filters and then tried on the real robot. As comparison, a simple PID controller with two states was implemented as well.

VIII. Results

A. Simulink

In figures 7 and 9 the real angle and angular velocity are shown using LQR with a complementary filter or a Kalman filter. In figures 8 and 10, the real angle, measured angle and filtered angle are shown for the different filters. The LQR weights used in the plots are $Q = diag(1000, 0.1, 5, 0.1)$ $R = 2$ and $N = diag(0, 0, 0, 0)$ (Q contains weights on the states, R on the input and N the combinations). This gives the LQR parameters: $\begin{bmatrix} -18.3341 & -0.0459 & -2.8688 & -0.0619 \end{bmatrix}$. The parameters for the Kalman filter are given in table II.

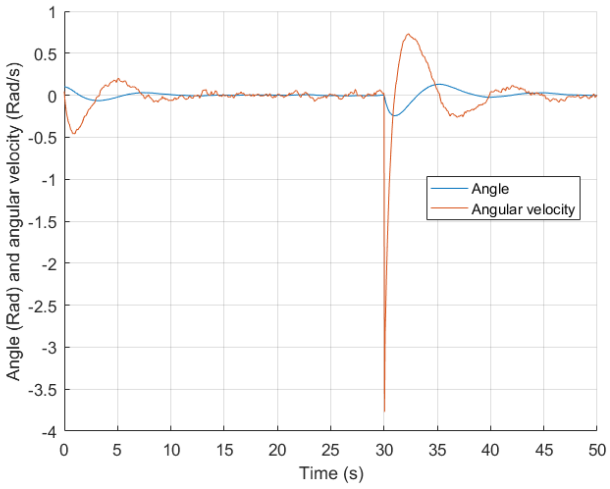

Fig. 7. Real angle and angular velocity using the complementary filter. There is an initial angle of 0.1 Rad, a step disturbance with amplitude 5 at time 30 s, and noise with variance 0.001 on the measured angle.

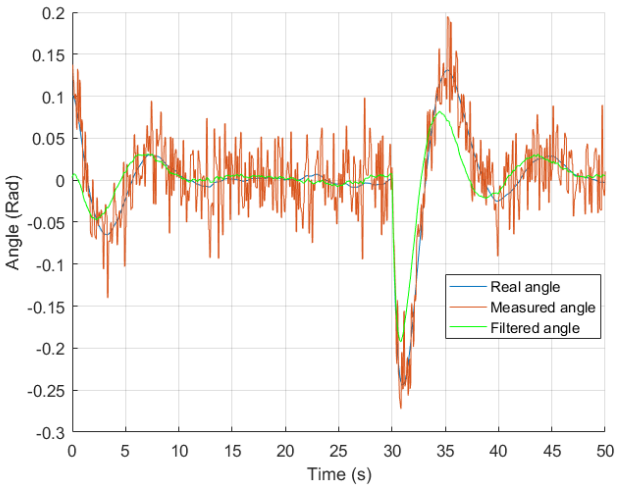

Fig. 8. Real angle, measured angle and filtered angle with complementary filter. There is an initial angle of 0.1 Rad, a step disturbance with amplitude 5 at time 30 s, and noise with variance 0.001 on the measured angle.

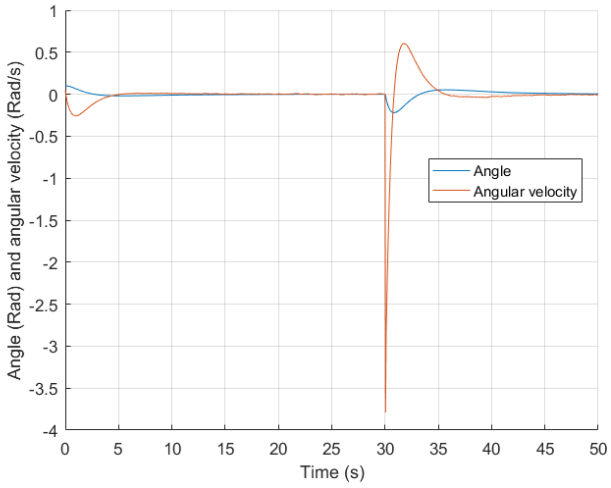

Fig. 9. Real angle and angular velocity using the Kalman filter. There is an initial angle of 0.1 Rad, a step disturbance with amplitude 5 at time 30 s, and noise with variance 0.001 on the measured angle.

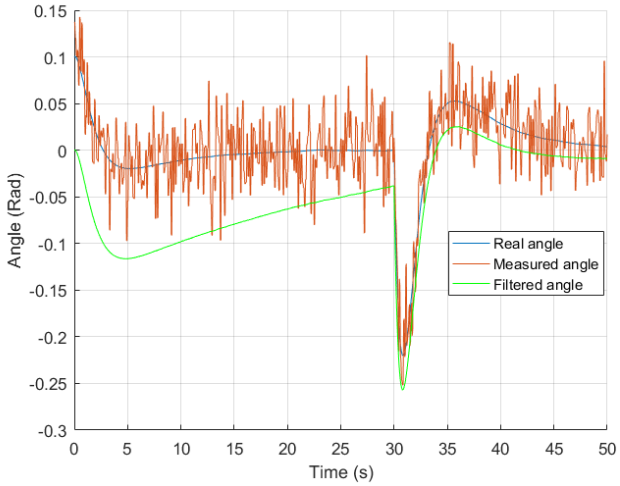

Fig. 10. Real angle, measured angle and filtered angle with Kalman filter. There is an initial angle of 0.1 Rad, a step disturbance with amplitude 5 at time 30 s, and noise with variance 0.001 on the measured angle.

B. The real system

In figure 11 the estimated angle from complementary filter, gyroscope and accelerometer is shown on the actual robot with an external force added to measure the performance of the system. Figure 12 the estimated angle from Kalman filter, gyroscope and accelerometer is shown on the actual robot with an external force added to measure the performance of the system. In table I the measured constants of the robot are shown, table III shows our LQR constants for the robot and in table II the Kalman constants are shown.

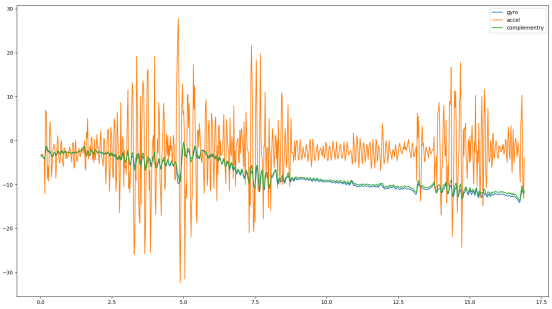

Fig. 11. LQR: Estimated angle from the Complementary filter, Gyroscope and accelerometer at different times. An external force was introduced to the system after 5 seconds

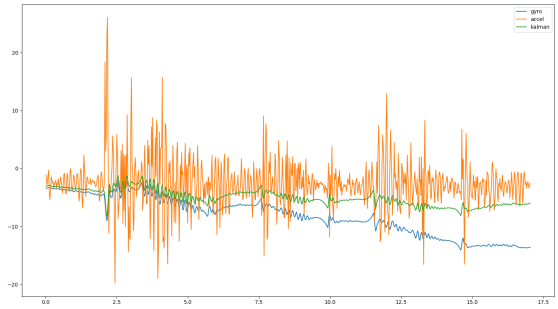

Fig. 12. LQR: Estimated angle from the Kalman filter, Gyroscope and accelerometer at different times. An external force was introduced to the system at around 2.5 seconds

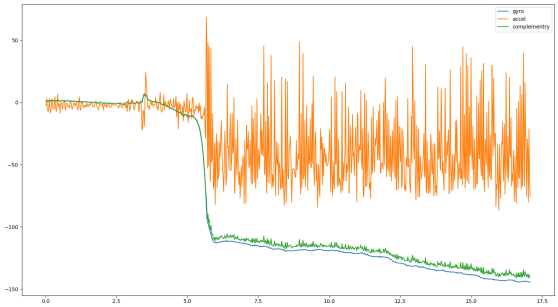

Fig. 13. PID: Estimated angle from the complementary filter, Gyroscope and accelerometer at different times. An external force was introduced to the system at around 2.5 seconds

TABLE I
MEASURED CONSTANTS

m	$0.554kg$
M	$0.060kg$
r	$0.04m$
l	$0.11m$
g	$9.82m/s^2$
I_w	$9.6 \cdot 10^{-5} kg \cdot m^2$
I_b	$0.0067kg \cdot m^2$

TABLE II
KALMAN VALUES

R_v	43
Q_{accel}	0.03
Q_{gyro}	0.00000002

TABLE III
LQR-VALUES

k_θ	18.1696
k_{θ_w}	0.08
$k_{\dot{\theta}}$	0.85
$k_{\dot{\theta}_w}$	0.083

TABLE IV
PID VALUES

K_p	19
K_i	1.82
K_d	24.5

IX. DISCUSSION

Based on figure 11, the complementary filter starts to drift away from the actual value. This was expected since the filter is based mostly on the values from the gyroscope due to the variance of the accelerometer values, which can be seen in equation 1. By using our LQR controller with the complementary filter the robot managed to self balance. However, due to the drift in the motors, the position of the robot gets further away from the starting point. Using the Kalman filter, this drift was removed, which can be seen in figure 12. The values for the Kalman filter that were used on the robot are shown in table II. A high value of the measurement noise was chosen since the accelerometer angle tends to be corrupted by noise and the angle from the gyroscope drifts. The Q values show how much we trust the sensor outputs (see section III-B for more details) compared to the other, where a high value indicates that the sensor was trusted less compared to the other sensor. Based on the result from table II, the accelerometer was trusted less than the gyroscope which worked for the system. The values from table III are the results from trial and error to calculate the Q matrix in section III-C which balanced the robot. With the PID implementation the robot was successfully balanced. However, if external force was added on the robot it was not successful in reducing the disturbance resulting in the robot oscillating and eventually tipping over, which can be seen in figure 13.

Comparing the results from the different controllers, they suggest that an LQR controller is better than the PID to self balance a robot. However, it might be that the PID is not properly tuned. It is also worth noting that the EV3 with leJOS had a limitation in the speed of the control loop. The minimum time per loop was around 16 ms, and it was not possible to go lower than that if the values from the accelerometer and gyroscope were to be collected. It is possible that a PID controller would work better with a lower downtime per iteration. One could try doing the control strategies in C or the EV3 language, which could help with reducing the delay. Furthermore, it is possible that both controllers could have worked better if 4 motors were used instead of 2 so the robot could recover balance from steeper angles.

As for the filters it looks like the Kalman filter performed better than the complementary filter to estimate the angle. That being said, the implementation has not taken care of the bias offset from the gyroscope which might be what makes the complementary filter follow the gyroscope so closely. So if one takes care of the bias offset from the gyroscope and then use the complementary filter the result might be closer to the Kalman filter.

REFERENCES

[1] Bong Seok Park Byung Woo Kim. *Robust Control for the Segway with Unknown Control Coefficient and Model Uncertainties*. June 2016. URL: http://dx.doi.org/10.3390/s16071000.

[2] Rolf Johansson. *Predictive and Adaptive Control*. 2015.

[3] Kristian Sloth Lauszus. *A practical approach to Kalman filter and how to implement it*. Sept. 10, 2012. URL: http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/.

[4] Lauren Valk. *Tutorial: Building BALANC3R*. June 23, 2014. URL: http://robotsquare.com/2014/06/23/tutorial-building-balanc3r/.