Two-Wheeled Balancing Robot

Mads Bornebusch (s123627), Kristian Lauszus (s123808)
31015 Introductory Project - Technical University of Denmark, DTU

June 24, 2014

***Abstract* - This paper describes the work of designing and building a two-wheeled self-balancing robot capable of carrying a person. This system is similar to the non-linear and unstable inverted pendulum on a cart. The linearised system equations for the robot dynamics is found and the system is simulated and stabilised with a controller in Matlab. The robot is built using two 24V 500W DC motors. The robot is controlled with an Arduino Pro mini with an 8-bit ATmega328P AVR microcontroller running at 16MHz. To find the angle of the robot the readings from a MPU-6050 gyroscope and accelerometer is fused using a Kalman filter. The controller we have implemented is a manually tuned PID-controller written in C/C++. With a Bluetooth module we were able to establish communication between the robot and an Android Application written for the purpose so that data could be read from the robot and the PID-values of the controller could be set. By manual tuning it was possible to achieve stability for the robot carrying a person.**

I. Introduction

In today's modern world there are several different types of personal transportation. One of the more interesting ones is the two-wheeled balancing robot, commercially known as a Segway. It seems fascinating that this vehicle is able to keep upright while a person is driving it. There is however room for improvement of this vehicle by making data such as speed and a map with location available to the user while driving. This paper will cover the process of building and programming a balancing robot which can communicate with an Android application via Bluetooth.

II. Model of the system

There are several different ways to model a balancing robot. One of the simplest ways is to model it as a cart with an inverted pendulum on top. As our robot is built with two wheels directly mounted on the platform on which the user stand, the torque form the motors will have an effect on both the wheels and the body of the robot. The robot will therefore be modelled as two wheels connected to a body. The free body diagram of this model is shown on Fig.1. The following assumptions are used when modelling the robot:

- The wheels are rolling (no slipping)
- All bodies are rigid
- The body is rotating around the center of gravity (CG)
- The mechanical losses are approximately zero

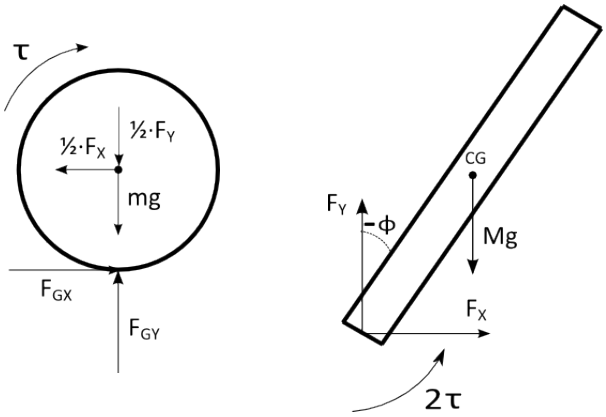

Fig. 1: The free body diagrams for one of the wheels (left) and the body (right)

When using Newtons second law of motion and the relationship between torque and angular acceleration, a set of equations for the wheel and the body of the robot (including the driver) can be set up. The forces from the free body diagram in Fig.1 is used. The equations for each wheel are:

$$m \cdot \ddot{x}_w = F_{GX} - \frac{1}{2} \cdot F_X \tag{1}$$

$$0 = F_{GY} - \frac{1}{2} \cdot F_Y - m \cdot g \tag{2}$$

$$I_w \cdot \ddot{\theta} = -\tau + F_{GX} \cdot r \tag{3}$$

The equations for the x- and y-direction of the body can easily be written. For the torque equation, we have to translate the torque τ at the end of the body to a torque in the CG. As the angle that the body is rotating is the same whether it is rotating around the end or the CG we say that $\frac{\tau_{CG}}{I_{CG}} = \frac{\tau_{end}}{I_{end}}$ which will yield $\tau_{CG} = \frac{I_{CG}}{I_{end}} \tau_{end}$. The equations for the body can now be set up:

$$M \cdot \ddot{x}_b = F_X \tag{4}$$

$$M \cdot \ddot{y}_b = F_Y - M \cdot y \tag{5}$$

$$I_{b,CG} \cdot \ddot{\theta} = 2\frac{I_{b,CG}}{I_{b,end}} \tau - F_Y \sin\phi \cdot l + F_X \cos\phi \cdot l \tag{6}$$

An approximation when the robot is balancing is that $x_b = x_w$. The y-coordinate will be constant for both the

wheel and the body of the robot. This will give $\ddot{y}_B = 0$. These approximation can be substituted in equation (4) and (5). The forces F_X, F_Y, F_{GX} and F_{GY} can now be eliminated from the equations (1) to (6) which will give the two governing equations for the motion of the robot. These equations are linearised with the small angle approximations $\sin\phi = \phi$ and $\cos\phi = 1$ and with the assumption that the angular velocity i sufficiently small to be approximated to zero: $\dot{\phi}^2 = 0$. The linear equations are:

$$(2I_w - Mr^2 - 2mr^2)\ddot{x}_w = -2\tau r \tag{7}$$

$$I_{b,CG}\ddot{\phi}I_{b,end} = M\ddot{x}_w l I_{b,end} - Mgl\phi I_{b,end} + 2\tau I_{b,CG} \tag{8}$$

From these equations the transfer functions from τ to ϕ and from τ to $\ddot{x}_w$ can be found by eliminating a variable and Laplace transforming. The transfer function from τ to X is shown in (9). In a similar way, the transfer function from τ to ϕ, $G_{\tau\phi} = \frac{\phi}{\tau}$ can be found. This transfer function is shown in (10).

$$G_{\tau X} = \frac{-2r}{s^2(-Mr^2 - 2mr^2 + 2I_w)} \tag{9}$$

III. Simulation

The transfer functions for the robot was used to simulate it and to design a controller in Matlab. As the model does not include the dynamics of the motors, the controller designed in Matlab can not be implemented directly on the robot. The purpose of these simulations is to show that stability of the system can be achieved. When simulating the system it was additionally assumed that the body of the robot (including the driver) can be modelled as a rod. The measured data for the simulation is shown in Table 1.

Table 1: Measured quantities for the simulations

Mass of wheel	m	5.15 kg
Mass of robot without wheels	m_{robot}	20.1 kg
Mass of person	m_{person}	80 kg
Height of person	h	1.83 m
Radius of the wheel	r	0.17 m
Radius of the rim	r_{rim}	0.097 m
Cross sectional torus radius	b	0.0385 m

From these measured quantities, the rest of the quantities for the transfer function can be calculated. The moment of inertia for the a rod rotating about the end and about the CG is $I_{b,end} = \frac{Mh^2}{3}$ and $I_{b,CG} = \frac{Mh^2}{12}$. The wheel is split up in the rim and the tire. The moment of inertia for the tire is that for a torus ($(a^2 + \frac{3}{4}b^2)m_{rubber}$) [12]. The mass of the torus is the volume of it ($2\pi^2 b^2(r_{rim} + b)$) times the density of rubber (which is approximately $1.1 \cdot 10^3\ kg/m^3$ [5, p. 11]). The moment of inertia for the rim is that of a disk having a mass that is $m - m_{rubber}$. The total moment of inertia for the wheel is the sum of that for the torus and the disk. The quantities used in the simulations are shown in Table 2.

Table 2: Calculated quantities for the simulations

m	$5.15\ kg$
M	$100.1\ kg$
l	$0.915\ m$
r	$0.17\ m$
$I_{b,end}$	$111.74\ kgm^2$
$I_{b,CG}$	$27.94\ kgm^2$
I_w	$0.089\ kgm^2$
g	$9.8\ m/s^2$

A. Controller design with Ziegler-Nichols method

To analyse the system a Bode plot was used and the poles of the system was found. The Bode plot is shown in Fig. 2.

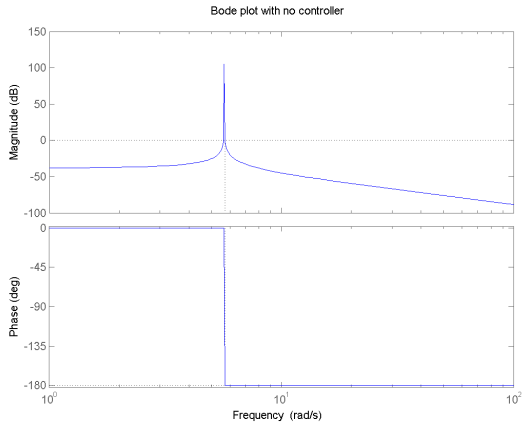

Fig. 2: Bode plot of the system with no controller

The system has two poles on the imaginary axis at $\pm 5.668i$. This is also visible at the Bode plot where the phase "jumps" $-180°$ at approximately 5.7rad/s. The gain is chosen to be $K_{P,crit} = 800$ to get the magnitude above 0dB until approximately 20rad/s. The cycle time is measured to $T_0 = 0.4s$ and the controller is now designed after the Ziegler-Nichols method [10, p. 296]. The formulas and the values are shown in Table 3.

Table 3: Values for the Ziegler-Nichols designed controller

Formula	Value
$K_P = 0.6K_{P,crit}$	480
$\tau_i = 0.5T_0$	0.2
$\tau_d = 0.13T_0$	0.052

The transfer function of the controller that is implemented is the one shown in (11) with $\alpha = 0.1$

$$G_c(s) = K_P \frac{\tau_i s + 1}{\tau_i s} \frac{\tau_d s + 1}{\alpha\tau_d s + 1} \tag{11}$$

This transfer function is that of a PI-Lead controller which is how the theoretical PID-controller can be implemented.
With the controller in the system, the step response will be as shown on Fig. 3.

$$G_{\tau\phi} = \frac{2(-I_{b,CG}Mr^2 - 2I_{b,CG}mr^2 - I_{b,end}Mlr + 2I_{b,CG}I_w)}{I_{b,end}((-I_{b,CG}Mr^2 - 2I_{b,CG}mr^2 + 2I_{b,CG}I_w)s^2 - M^2glr^2 - 2Mglmr^2 + 2I_wMgl)} \tag{10}$$

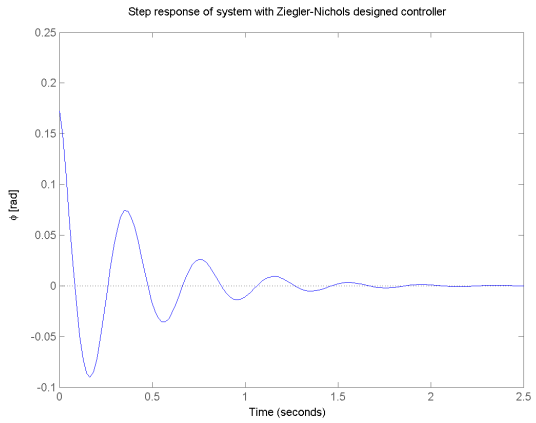

Fig. 3: Step response of system with the Ziegler-Nichols designed controller. The step is $10°$ on ϕ

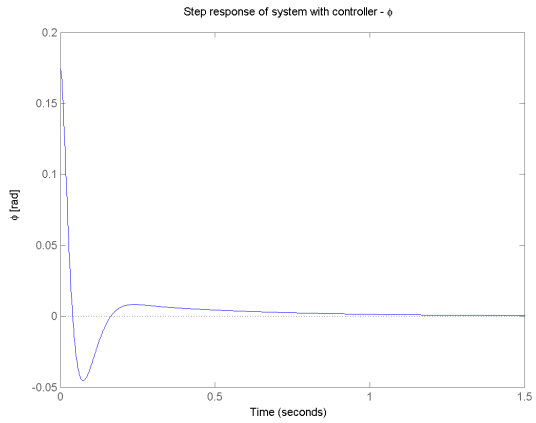

Fig. 4: Step response of the system with the controller designed by choosing ω_c. The step is $10°$ on ϕ

The step response is oscillatory but damped and the figure shows that the output is almost stable 2s after the input was applied. So it can be seen that it is possible to make the system stable. The response will be even better if the system is made slower (the cycle time T_0 is increased).

B. Controller design by choosing ω_c

Another way of designing a controller is by choosing the cross frequency, ω_c. When this frequency is chosen, the values for the controller can be calculated with the equations in Table 4. We have chosen ω_c to be 40rad/s.

Table 4: Values for controller designed by placing ω_c

Formula	Value
$K_P = \frac{1}{\lvert G_C(\omega_c)G_O(\omega_c)\rvert}$	1275
$\tau_i = 5\tau_d$	0.3953
$\tau_d = \frac{1}{(\omega_c\sqrt{(\alpha)})}$	0.0791

This design method will give a better step response from the system. It is shown in Fig.4.

The oscillations are damped and the system is in almost perfect equilibrium 1 second after the disturbance is applied. The system is sufficiently fast to be usable. In order to be that fast the system needs to apply a very large torque. The torque at time 0 is 2230Nm which is not realistic with the used motors. Assuming that the motor is a DC shunt-connected motor, the equation $\tau_{dev} = \frac{K\phi}{R_A}(V_T - K\phi\omega_m)$ will apply [1, p. 803]. The motor have a no load speed at 24.1V, 2953rpm and a rated load 1.89Nm at 24.35V, 2542rpm [9]. From these numbers the maximum torque (with a gearing of 6.7:1) is calculated to 84.3Nm for each motor.

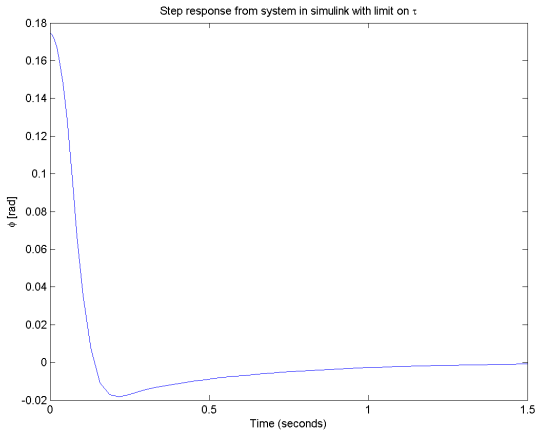

Fig. 5: Step response of the system with the controller designed by choosing ω_c. The step is $10°$ on ϕ and τ is saturated at 84.3Nm

In Simulink it is possible to put a saturation in the system. When doing this the step response will be as shown on 5. The figure shows that the system is still stable even with the saturation of τ. This shows that it is indeed possible to make the system stable when using a PI-Lead controller.

IV. Hardware

The robot is controlled by an 8-bit ATmega328P AVR microcontroller runnning at 16MHz. The microcontroller takes measurements from a MPU-6050 via I2C. This is a so called IMU (Inertial measurement unit) which in this case is a 3-axis accelerometer and a 3-axis gyroscope in one package.

The angle is calculated by taking atan2 to the y- and z-components from the accelerometer[4]. The angle obtained in this way will have a lot of high frequency noise on top of

it due to linear accelereration when the robot moves. Therefore the gyroscope is needed. The output from the gyroscope is converted to degrees per seconds and then integrated to obtain the angle. This approach will also integrate the error which will make the angle drift over time. To solve these issues we are using a Kalman filter written by Kristian Lauszus[8]. The Kalman filter takes the angle obtained from the accelerometer, the gyroscope reading in degrees per second and Δt as arguments. The angle from the Kalman filter is then used as input to a PID-controller which will output the PWM-value to the motor driver controlling the motors.

The schematic in Fig. 6 shows the circuit of the robot. It consist of an Arduino Pro Mini, a 7805 linear 5V regulator a LM324 operational amplifier which is used as a buffer for a deadman switch - see section A. Safety concerns. The LM324 is also used to buffer the output from a voltage divider connected directly to the 6S LiPo batteries for measuring the voltage. The robot is powered by three of those batteries, giving a total capacity of 9000mAh.

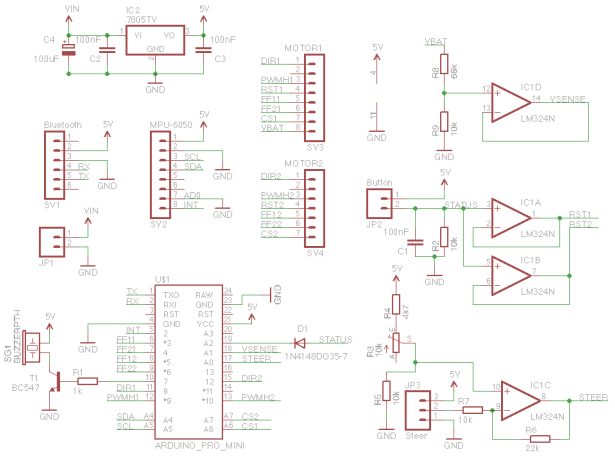

Fig. 6: Schematic of the main PCB

The last operational amplifier is used as a inverting amplifier. The negative input is connected to a 10kΩ potentiometer through a 10kΩ resistor. The potentiometer is connected to the steering mechanism. The positive input is connected to an adjustable voltage divider. This allows us to adjust the output to roughly 2.5V when the steering mechanism is in the neutral position. This does not have to be exactly 2.5V, as the microcontroller will calibrate the steering when powered on. This allows us to tell if the user is tilting the steering rod and by how much. This reading is used in the control algorithm to make the robot turn.

The steering mechanism is made out of $1/2''$ water pipe. It consists of the handlebar at the top mounted to 105cm of vertical pipe which is connected to a horizontal 30cm piece using a 90° fitting. The horizontal piece is mounted inside two bearings with the potentiometer mounted in the end of it. A solid piece of iron was welded to the inside of the pipe at the 90° fitting to give some extra strength. Finally two springs are mounted to give mechanical feedback when moving the steering rod.

On the main PCB we have broken out headers for the MPU-6050, a Bluetooth SPP module[3] and the two motor drivers[11]. These are rated a 34V at a continuous current of 23A. We have mounted these on some custom heatsinks which should increase the rated continuous current to well above 30A. The motor drivers are used to drive the 500W 24V DC motors[9]. A custom hub is mounted between each of the motors and the rim of a 3.00-8 solid rubber wheel.

The entire board is powered using a LM2596 adjustable switching regulator. This is used to step the voltage from the batteries (25.2V to 21.6V) down to approximately 8V. Finally a buzzer is connected to the microcontroller via a BC547 transistor to give audio feedback to the user.

Fig. 7 shows a picture of the final robot with all the electronics mounted on the top on a breadboard. The vertical metal piece below the robot is just there to keep it upright while taking the photo.

Fig. 7: Picture of the finished robot

A. Safety concerns

To prevent the robot from applying full power to the motors due to some bug in the firmware we decided to use a deadman button which is a momentary push button on the handlebar. It is connected to two operational amplifiers configured as buffers - see Fig. 6. The outputs from the two buffers are connected to the reset lines of the motor drivers. Ensuring that the motor drivers are reset unless the button is pressed.

Furthermore some safety precautions should be taken with regards to the LiPo batteries, as they can potentially catch on fire if not treated correctly. Therefore it is strictly critical that all the cells are balanced and they should not be mechanically damaged.

V. Code for the robot

The code for the robot is written in C/C++. It uses some functions and libraries from the popular open source platform Arduino. This includes the I2C library (called Wire) and the

time keeping functions micros and millis[2]. Furthermore it uses a Kalman filter library written by Kristian Lauszus[8].

Fig. 8 shows a rough flow chart of the code running on the microcontroller. At first the microcontroller is powered on by connecting the battery to the LM2596 regulator. Then it initializes an input which makes it able to read the current status of the deadman button. After that it sets the pin connected to the buzzer to output and reads the different values from the EEPROM including PID-values, Kalman values etc. If it detects that the EEPROM values should be restored (if the structure of the values have changed due to a value being added or removed) it will turn on the buzzer for 1 second and proceed afterwards. The next step is to initialize the UART (Universal asynchronous receiver/transmitter) at 57600 baud rate, the pins connected to the motor drivers and finally configure the MPU-6050 at a sample rate of 500Hz and set the accelerometer $\pm$2g and the gyroscope range to $\pm$250deg/s. Then calibrates the steering and finally it turns on the buzzer for 100 milliseconds and initializes the timers it uses for timekeeping.

Now the main loop is running. As it can be seen on the flow chart it will run forever until power is removed. The first step is to check if any of the diagnostic pins on the motor driver are high, this means that there is some kind of fault[11]. If that is the case it will turn on the buzzer to alert the user that something is wrong. The next is to check if any new IMU data is available. The MPU-6050 is configured so it will hold a pin high if there are new measurements available. This is needed, as the microcontroller can run the loop much faster than the sensors sample rate. If there is no new data from the IMU it will check if any data is available at the serial port. This is used to configure PID-values, Kalman values, the amount of turning etc. If new measurements are available it will read the sensors and calculate the angle using the Kalman filter. Then it reads the steering input from the potentiometer and checks if the robot is upright and the deadman button is pressed. If that is not the case it either means that the robot has fallen over or the user has let go of the button. In this case it will turn off the motors. If not it will calculate new PWM-values for the motors using a PID-controller.

Finally the battery voltage is checked. If the battery is too low it will start the buzzer. After this final step it will go back to the start of the loop.

One important aspect of the PID-controller is that it will try to push the user back when the robot speeds up. This will make sure that the motors will never reach their maximum speed which would cause the robot to fall over. Furthermore the turning speed is decreased as the speed of the motors increases. This is needed as the gentle turning at high speeds and rapid turning when standing still will make the user less likely to fall off. A rough estimate of the speed of the motors is done by integrating the PWM-value.

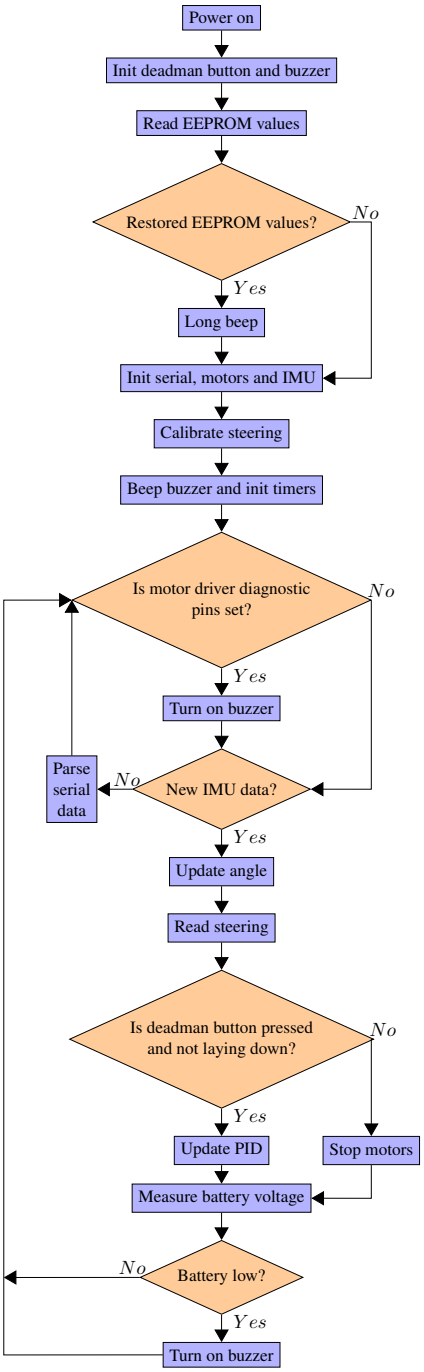

Fig. 8: Flowchart of the main loop

VI. Android Application

In order to easily tune the PID-values and get data from the robot we wrote an Android application in Java. A picture of the different screens can be seen at Fig. 9.

In the first screen the user is able to see the current draw of the motors, the turning value, battery level, run time and finally the PWM-value on a custom speedometer. The next screen allows the user to adjust the PID-values, target angle and adjust how fast the robot should be able to turn. The third screen shows the current position on a map. The posi-

tion is obtained using the built-in GPS module of the Android device. The fourth and final screen shows a graph of the current angle calculated using the accelerometer, gyroscope and the angle estimated using the Kalman filter. It also allows the users to adjust the Kalman filter coefficients.

The communication between the microcontroller and Android application is done via Bluetooth. In order to make sure that the data is parsed properly a protocol for this was implemented. It consist of a constant string header, then one byte to indicate the command and then one byte indicating the length of the data to follow. After the data there is a checksum. This checksum is calculated by taking XOR of each byte in the message excluding the header. The full serial protocol can be seen in the source code[7].

Fig. 9: Screenshot of Android application

VII. Discussion and improvements

The final product turned out just as we planned, but there is still room for some improvements that we left out due to time constraints. One major thing would be to mount encoders on the robot, so the velocity could be estimated more precisely. This should make it possible to get the robot to travel a bit smoother especially uphill. It would also allow it to balance on its own without a person on it. Also we would consider routing out a slot in the plate for the steering rod to prevent the bottom of the steering mechanism from hitting the ground if the user falls off. The pipe have already been reinforced by welding a solid piece of iron inside the pipe at the 90° fitting in order to make it able to withstand falling. Also the PID-controller could be improved a bit as we could implement dynamic gain that would depend on the battery level and weight of the person riding the vehicle. This could for instance be measured using strain gauges mounted on the base. This would also allow us to tell if a person is standing on the robot or not. Also we have considered replacing the current tires (made out of solid rubber) with pneumatic tires to make them slightly lighter, but also to make the ride more comfortable on rough terrain.

VIII. Conclusion

We achieved stability of the robot carrying a person with manual tuning of the PID-values. The robot is able to send current draw, turning value, battery level, run time, PWM-value and data from the accelerometer, gyro and Kalman filter to the Android application. The application can set the PID-values, target angle, turning scale and Kalman filter values. The project was thus successful in terms of achieving stability of the robot and communicating with an Android Application. Further improvements on the the design can be made such as improving the PID-control algorithm and adding encoders to the wheels to measure the speed and position of the robot.

References

[1] Allan R. Hambley. (2011). *Electrical Engineering.* Pearson

[2] Arduino (2014) Arduino Language Reference. *http://arduino.cc/en/Reference/HomePage*

[3] Bluetooth Developer Portal (2014). Serial Port Profile (SPP). *https://developer.bluetooth.org/TechnologyOverview/Pages/SPP.aspx*

[4] Freescale Semiconductor (2013) Tilt Sensing Using a Three-Axis Accelerometer. *http://www.freescale.com/files/sensors/doc/app_note/AN3461.pdf*

[5] Geosyntec Consultants (2008) Guidance Manual for Engineering Uses of Scrap Tires *http://www.mde.state.md.us/assets/document/Guidance_Manual_For_Scrap_Tires.pdf*

[6] Github, BalancingRobotFullSize (2014) Source code for the robot. *https://github.com/Lauszus/BalancingRobotFullSize*

[7] Github, BalancingRobotFullSizeAndroid (2014) Source code for Android app. *https://github.com/Lauszus/BalancingRobotFullSizeAndroid*

[8] Kristian Sloth Lauszus, TKJ Electronics (2012) A practical approach to Kalman filter and how to implement it. *http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/*

[9] Mat-Con (2014). Spur gearing electric motor Unite MY1020Z2 DC 500W 24V 12,6Nm. *http://shop.mat-con.net/epages/62158737.sf/en_GB/?ObjectPath=/Shops/62158737/Products/my1020z2*

[10] Ole Jannerup, Poul Haase Sørensen. (2009). *Reguleringsteknik.* Polyteknisk Forlag

[11] Pololu (2014). Pololu High-Power Motor Driver 24v23 CS. *http://www.pololu.com/product/1456*

[12] Wikipedia (2014) List of moments of inertia *http://en.wikipedia.org/wiki/List_of_moments_of_inertia*