Two wheeled balancing LEGO robot

Jonsson Per, Piltan Ali, Rosén Olov

September 17, 2009

Abstract

A two wheel inverted pendulum robot have been constructed using the building bocks from LEGO mindstorms kit. A 3-DOF mathematical model describing the motion of the robot has been derived using Lagrangian dynamics. The objective of this project were to design controllers for balancing and line tracking. Two regulators were designed using the mathematical model, one PID regulator and one LQG regulator. Software for development and simulation of the regulators were created using Matlab and its toolbox "Control Toolbox". These regulators were then implemented and evaluated on the real system using RobotC. Both the PID and LQG regulator successfully balanced the robot and the LQG regulator were also able to perform servo regulation, where the reference signals to the robot were given using a gamepad connected to a computer.

Also an algorithm for line tracking using a light sensor attached to the front of the robot were implemented and tested. Using the algorithm the robot was able to follow an eight shaped track with good performance. But the algorithm could definitely be improved to make better lap times.

Contents

1 The robot

1.1 Modeling of the robot

When designing controllers a good mathemathical model for the robot is needed for several reasons. For example it is much easier to test different controllers on a model using a simulation enivronmeant instead of trying them on the robot. In this way we can design and analyze the controller on the model before implementation. Also most control design methods are model based and this means that the model will affect the performance of the designed controllers, thus a good model is essential. There are several articles published on the modeling of the two wheeled inverted pendulum,[1, 2, 3], where the mathematical model is given. We choose however to derive the model on our own. By doing so we could verify the existing models with ours and gain insight to the problem. Here we show the important steps of the model derivation.

The robot is modelled as shown in figure 1 and figure 2. The proposed model describes the 3-DOF dynamics of the robot and is derived using Euler/Lagrange's method. The Lagrangian expresses the difference between the kinetic energy and potential energy of the robot, $L = T - U$. In order to express the kinetic energy for the robot three generalized velocities is needed, the linear velocity v, the angular pitch velocity $\dot{\psi}$ and the jaw angular velocity $\dot{\phi}$. The generalized velocities can be chosen in many ways, here we have decided for these velocities because they correspond to the variables we wish to control.

The Euler/Lagrange equations of motion for the robot are given by

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i \qquad i = 1, 2, ..., n \tag{1}$$

Where $q = (x,\, \phi,\, \psi)$ are the generalized coordinates linear position, jaw angle, and pitch angle and $\dot{q}$ are the corresponding generalized velocities. Q_i denotes the generalized force for each coordinate.

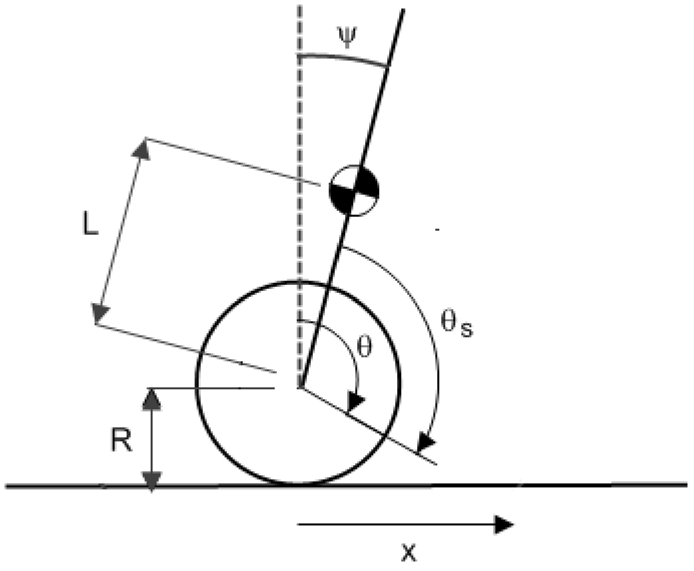


Figure 1: Side view of the robot.

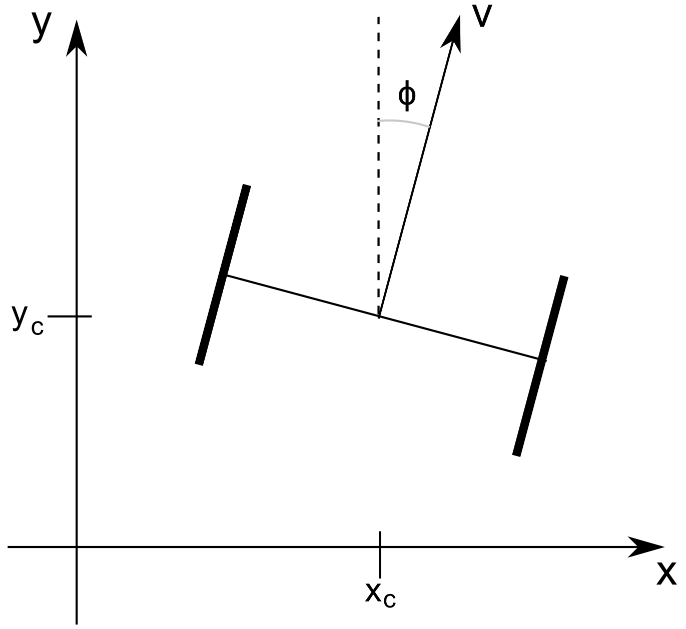


Figure 2: Robot viewed from above

The physical parameters of the robot are the following.

g	$[\frac{m}{s^2}]$	:Gravitational acceleration constant
m_b	$[kg]$	:Body mass
m_w	$[kg]$	:Wheel mass
R	$[m]$	:Wheel radius
L	$[m]$	:Distance to center of gravity from wheel axle
W	$[m]$	:Half body width
I_p	$[kgm^2]$	:Body pitch inertia moment
I_j	$[kgm^2]$	:Body jaw inertia moment
I_{xx}	$[kgm^2]$	:Body inertia moment about x axis
I_w	$[kgm^2]$	:Wheel inertia moment
J_m	$[kgm^2]$	:DC motor rotor inertia moment
R_a	$[\Omega]$	:DC motor resistance
K_b	$[V\frac{s}{rad}]$	:DC motor back EMF constant
K_t	$\frac{Nm}{A}$	:DC motor torque constant
B_m	Nms	:Damping ratio of the mechanical system

The kinetic energy of the body, T_b, is given as

$$T_b = \frac{m_b}{2}[\dot{x}^2+2L\dot{\psi}\cos\psi\dot{x}+L^2\dot{\psi}^2+L^2\dot{\phi}^2\sin^2\psi]+\frac{1}{2}[I_{xx}\dot{\phi}^2\sin^2\psi+I_{yy}\dot{\psi}^2+I_{zz}\dot{\phi}^2\cos^2\psi] \tag{2}$$

And the kinetic energy for the wheels, T_w, is expressed as

$$T_w = \frac{1}{2}m_wR^2(\dot{\theta_r}^2+\dot{\theta_l}^2)+\frac{1}{2}I_w(\dot{\theta}_r^2+\dot{\theta_l}^2) = (m_w+\frac{I_w}{R^2})(\dot{x}^2+W^2\dot{\phi}^2) \tag{3}$$

The potential energy for the robot is just the potential energy for the CoG of the body which is

$$U = m_bgL\cos\psi \tag{4}$$

Then the expression for the Lagrangian is

$$L = \left[\frac{m_b}{2}+m_w+\frac{I_w}{R^2}\right]\dot{x}^2+\left[\frac{m_bL^2}{2}+\frac{1}{2}I_p\right]\dot{\psi}^2+\left[(m_w+\frac{I_w}{R^2})W^2+\frac{1}{2}I_j\cos^2\psi\right.$$
$$\left. ...+\frac{1}{2}I_{xx}\sin^2\psi+\frac{1}{2}m_bL^2\sin^2\psi\right]\phi+m_bL\cos\psi\dot{x}\dot{\psi}-m_bgL\cos\psi \tag{5}$$

Now we evaluate (1) for each of the coordinates.

x - coordinate

$$\left[m_b + 2m_w + 2\frac{I_w}{R^2}\right]\ddot{x} + m_b L\cos(\psi)\ddot{\psi} - m_b L\sin(\psi)\dot{\psi}^2 = \frac{(M_{right} + M_{left})}{R} \quad (6)$$

ϕ - coordinate

$$\left[2(m_w + \frac{I_w}{R^2})W^2 + I_{xx}\sin^2(\psi) + I_J cos^2(\psi) + m_b L^2\sin^2(\psi)\right]\ddot{\phi} + ...$$
$$2\left[m_b L^2 + I_{xx} - I_J\right]\sin(\psi)\cos(\psi)\dot{\psi}\dot{\phi} = \frac{W}{R}(M_{right} - M_{left}) \quad (7)$$

ψ - coordinate

$$\left[m_b L^2 + I_p\right]\ddot{\psi} + m_b L\cos(\psi)\ddot{x} - \left[m_b L^2 + I_{xx} - I_J\right]\sin(\psi)\cos(\psi)\dot{\phi}^2 - ...$$
$$m_b g L\sin(\psi) = -(M_{right} + M_{left}) \quad (8)$$

where M_{right} and M_{left} are the torque exerted on right and left wheel by the DC-motor. Since we control motor voltages V_{right} and V_{left}, and not the torques M_{right} and M_{left} we must include the dynamics describing the relationship between these variables, this relation is given by

$$M_{right} + M_{left} = \frac{K_t}{R_a}(V_{right} + V_{left}) - 2\frac{J_m}{R}\ddot{x} - \frac{2}{R}(B_m + \frac{K_t K_b}{R_a})\dot{x}...$$
$$+ 2J_m\ddot{\psi} + 2(B_m + \frac{K_t K_b}{R_a})\dot{\psi} \quad (9)$$

$$M_{right} - M_{left} = \frac{K_t}{R_a}(V_{right} - V_{left}) - 2\frac{J_m W}{R}\ddot{\phi} - \frac{2W}{R}(B_m + \frac{K_t K_b}{R_a})\dot{\phi} \quad (10)$$

(Here we have neglected the effects of the inductance in motor circuit dynamics)

Inserting (9) into (6) and (8), and inserting (10) into (7) gives after reordering of the terms the following relation between input voltage and motion dynamics of the complete model.

x - coordinate

$$\left[m_b + 2m_w + 2\frac{(I_w + J_m)}{R^2}\right]\ddot{x} + \frac{2}{R^2}(B_m + \frac{K_t K_b}{R_a})\dot{x} + (m_b L\cos(\psi) - 2\frac{J_m}{R})\ddot{\psi} - ...$$
$$\frac{2}{R}(B_m + \frac{K_t K_b}{R_a})\dot{\psi} - m_b L\sin(\psi)\dot{\psi}^2 = \frac{K_t}{R R_a}(V_{right} + V_{left}) \quad (11)$$

ϕ - coordinate

$$\left[2(m_w + \frac{I_w + J_m}{R^2})W^2 + I_{xx}\sin^2(\psi) + I_J cos^2(\psi) + m_b L^2 \sin^2(\psi)\right]\ddot{\phi} + ...$$
$$2\left[\left[m_b L^2 + I_{xx} - I_J\right]\sin(\psi)\cos(\psi)\dot{\psi} + \frac{W^2}{R^2}(B_m + \frac{K_t K_b}{R_a})\right]\dot{\phi}...$$
$$= \frac{W}{R}\frac{K_t}{R_a}(V_{right} - V_{left}) \quad (12)$$

ψ - coordinate

$$\left[m_b L^2 + I_p + 2J_m\right]\ddot{\psi} + \left[m_b L\cos(\psi) - 2\frac{J_m}{R}\right]\ddot{x} + 2(B_m + \frac{K_t K_b}{R_a})\dot{\psi} - ...$$
$$\frac{2}{R}(B_m + \frac{K_t K_b}{R_a})\dot{x} - \left[m_b L^2 + I_{xx} - I_J\right]\sin(\psi)\cos(\psi)\dot{\phi}^2 - m_b g L\sin(\psi)...$$
$$= -\frac{K_t}{R_a}(V_{right} + V_{left}) \quad (13)$$

Equations (11)-(13) is our proposed non-linear model of the robot. For our purposes however it is also necessary to have a linear representation of the model. In our case this is done by using the small angle approximation which is a first order Taylor expansion around the equilibrium point, $\psi = 0$. Although this might seem like a brute approximation of the non-linear model it is actually quite accurate for angles up to about 15 degrees.

The linearization of equations (11)-(13) are represented by a state-space model as

$$\begin{array}{rcl} \dot{\mathbf{x}}(t) & = & \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \\ \mathbf{z}(t) & = & \mathbf{M}\mathbf{x}(t) \\ \mathbf{y}(t) & = & \mathbf{C}\mathbf{x}(t) \end{array} \quad (14)$$

where $\mathbf{x}(t) = (x,\ \phi,\ \psi,\ \dot{x},\ \dot{\phi},\ \dot{\psi})$, $\mathbf{z}(t)$ is the performance signal and $\mathbf{y}(t)$ is the measured plant output signal. And

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & a_{43} & a_{44} & 0 & a_{46} \\ 0 & 0 & 0 & 0 & a_{55} & 0 \\ 0 & 0 & a_{64} & a_{64} & 0 & a_{66} \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ b_1 & b_2 \\ b_3 & b_4 \\ b_5 & b_6 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} \frac{1}{R} & \frac{W}{R} & -1 & 0 & 0 & 0 \\ \frac{1}{R} & -\frac{W}{R} & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

For specification of the matrix elements see Appendix .

1.2 Hardware

1.2.1 Lego Mindstorms NXT

The “Lego Mindstorms NXT Kit” is a kit which contains a collection of Lego pieces, sensors, actuators and a small computer - the “NXT Brick”. Thanks to the brick, computer logic could easily be incorporated into ordinary Lego constructions. The heart of the “NXT Brick” is an ARM 32-bit microprocessor which can be programmed by the user.

To to able to compile and upload a program to the “NXT Brick” some sort of development tool is needed. There are several different ones to choose from. We have decided to choose a development tool called “RobotC”. “RobotC” contains a complete development environment together with its own programming language. The language is very similar to C, except for some minor differences. The reason why we chose “RobotC” is that it is very easy to work with and it has a lot of built-in features, like e.g. data logging, gamepad support, debugging, wireless Bluetooth communication, sensor drivers and so on.

1.2.2 Sensors and actuators

To be able to interact with the outside environment the “NXT Brick” can be equipped with different sensors and actuators. Some of them are included in the “Lego Mindstorms NXT kit” others have to be ordered separately.

Gyro sensor A gyro sensor can be used to measure the angular velocity that the gyro sensor senses. In this project a “Hitechnic NXT Gyro Sensor” [4] is used for measuring the pitch angular velocity of the robot. An issue that has to be taken in consideration, when reading the gyro sensor, is that the angular velocity readings are biased. One can observe this phenomenon by reading the raw sensor value under different ambient conditions and notice that the readings differ even though the sensor is not moving. In this project the issue is solved by subtracting the average over the gyro sensor values for about one second during start-up from all the upcoming readings. The bias is then assumed to be constant during the time the robot balances.

Light sensor The light sensor that are part of the "Lego Mindstorms NXT kit" [5] is used to get a number that is proportional to the sensed light intensity. This sensor is used to make it possible for the robot to follow a dark line. The sensors can be put in either "active" or "passive" mode. "Active" means that light will be emitted from the sensor. This will make the sensor readings less sensitive to varying ambient light conditions.

Motors and encoders Two dc motors [6] are used to produce the torques needed to balance the robot. They are both included in the "Lego Mindstorms NXT kit". The analog dc voltages necessary to drive the motors are generated by the "NXT Brick". For more information about the motors see [7] and [8].

Each motor also includes an encoder. The encoder's job is to measure the number of steps (in units of 1°) that the motor axle has turned. This information could then be used as an input to the controllers used for balancing the robot.

2 Control

This section presents the controllers we have designed in this project. We will give some brief theory for the different controller types that we have implemented (PID and LQG). For more detailed descriptions we recommend [9, 10]. A complete description of our regulators, simulation results are also presented.

Since we have only three measured signals but 6 states in the statespace model we must use an observer. We have choosen to use a Kalman filter for this purpouse, which is described below.

The state space model we are using in the design is the discrete version of (14)

$$\begin{aligned} \boldsymbol{x}(n+1) &= \boldsymbol{A}\boldsymbol{x}(n) + \mathbf{B}\mathbf{u}(n) + \mathbf{H}\mathbf{w}(n) \\ \mathbf{z}(n) &= \mathbf{M}\boldsymbol{x}(n) \\ \mathbf{y}(n) &= \mathbf{C}\mathbf{x}(n) + \mathbf{v}(n) \end{aligned} \tag{15}$$

Where A, B and C are the discretized A_c, B_c and C_c matrices. $w(t)$ are system disturbances and $v(t)$ are measurement disturbances.

2.1 Kalman filtering

2.1.1 Short about Kalman filtering

If not all states for a system could be directly measured a Kalman filter may be used to get estimates for these states. If the system disturbances $\boldsymbol{w}(t)$ and the measurement disturbances $\boldsymbol{v}(t)$ are white Gaussian noise and the system is linear, the Kalman filter is the optimal estimator in the sense that it minimizes the estimation error variance. If covariance/cross covariance matrices for $\boldsymbol{w}(t)$ and $v(t)$ are known and given by

$$\boldsymbol{R_1} = E[\boldsymbol{w}(t)\boldsymbol{w}(t)^T], \quad \boldsymbol{R_2} = E[\boldsymbol{v}(t)\boldsymbol{v}(t)^T] \quad \boldsymbol{R_{12}} = E[\boldsymbol{w}(t)\boldsymbol{v}(t)^T]$$

the the associated Ricati equation could be solved to obtain a Kalman gain K. On state space form the Kalman system could be written as

$$\hat{\boldsymbol{x}}(n+1) = \boldsymbol{A}\hat{\boldsymbol{x}}(n) + \boldsymbol{B}\boldsymbol{u}(n) + \boldsymbol{K}[\boldsymbol{C}\hat{\boldsymbol{x}}(n) - \boldsymbol{y}(n)] \tag{16}$$

Where $\hat{\boldsymbol{x}}$ is estimates of the states $\boldsymbol{x}$.

In most cases$\boldsymbol{R_1}$, $\boldsymbol{R_2}$ and $\boldsymbol{R_{12}}$ are not exactly known. The Kalman filter could still be constructed in a sensible manner by regarding $\boldsymbol{R_1}$, $\boldsymbol{R_2}$ and$\boldsymbol{R_{12}}$ as design variables. If one for instance believe that there are large model errors compared to measurement errors, $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ should be chosen accordingly, i.e. the elements of $\boldsymbol{R_1}$ should be large compared to the elements of $\boldsymbol{R_2}$.

If $\boldsymbol{R_2}$ is chosen small the Kalman filter will trust the measurements more and become faster to respond on changes in the measured signals but this will also make the estimations more sensitive to measurement noise. There is thus a trade off of making a "fast" filter and a noise insensitive filter. By studying the Kalman estimates from test runs on the real system one can adjust $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ so that this trade off becomes appropriate.

2.1.2 Our Kalman filter

In our system we have three measured signals, left shaft angle, right shaft angle and pitch angular velocity. Note that of these measurements it is only the pitch angular velocity that is an actual state in our model. In the design we have for simplicity assumed that $\boldsymbol{R_{12}} = 0$. Since we believe that we have large model errors compared to measurement errors we started with a large value of $\boldsymbol{R_1}$ compared to $\boldsymbol{R_2}$. Test runs on the real system showed that we must have a fast responding filter in order to be able to track the rapidly changing values of $\psi$and $\dot{\psi}$.

Discussion on the form of R_1 and R_2 The process noise we use in the design of the Kalman filter is modeled as an input signal disturbance, $H = B$, entering the system via the input signal. For our model this will have a direct effect on the derivatives of states representing derivatives of physical quantities (velocity, jaw angular velocity and pitch angular velocity). This way model disturbances acts on the system like forces affecting the states indirectly through the model. This is a more physically realistic modeling then adding disturbances directly to all states.

It must be mentioned that we tried the approach of adding disturbances to all states directly at the beginning of the Kalman filter design (i.e. $H = I$). This gave good results in simulation but on the real system the estimates directly diverged. We have not fully realized why this happened. Using this approach, of adding disturbance to all states, the obtained regulator system became unstable ($\boldsymbol{A} - \boldsymbol{BL} - \boldsymbol{KC}$ had poles outside the unit circle). Even though it should be possible to use an unstable regulator system, something went wrong on the real system. When modeling disturbances as input signal disturbances we managed to locate all poles for the combined Kalman and feedback gain system within the unit circle, which makes it impossible for the estimates to diverge and good estimates were obtained.

2.2 PID regulator

2.2.1 Short about the regulator

PID regulators consist of three "parts", one proportional part, one integrating part and one differentiating part. The integrating part will regulate out static errors but also make the system more oscillating, the differentiating part will damp the system. PID controllers are not model based and could often give good regulation of quite complex systems if tuned properly. In the frequency domain the regulator could be expressed as

$$\mathbf{U}(s) = K_p \left[\mathbf{E}(s) + \frac{1}{T_i s}\mathbf{E}(s) + T_d s \mathbf{E}(s) \right] \tag{17}$$

where U(s) is the control signal E(s) is the control error and the three different terms in the parenthesis is the proportional, integrating and differentiating part. The differentiating part as it stands is very sensitive to noise and is often replaced by a non ideal differentiation, i.e. $T_d s$ is replaced by

$$\frac{T_d s}{\alpha T_d s + 1}$$

here α is the parameter which decides how close to ideal differentiation one would like to get.

Also for implementation purposes we would like to represent the PID controller, in (17), on state space form which is given by,

$$\begin{aligned} \mathbf{u}(t) &= \begin{bmatrix} K_p & -K_p \end{bmatrix} \begin{bmatrix} \mathbf{x_i}(t) \\ \mathbf{x_d}(t) \end{bmatrix} + K_p(1 + \tfrac{1}{\alpha})\mathbf{e}(t) \\ \begin{bmatrix} \dot{\mathbf{x}}_\mathbf{i}(t) \\ \dot{\mathbf{x}}_\mathbf{d}(t) \end{bmatrix} &= \begin{bmatrix} 0 & 0 \\ 0 & -\frac{1}{\alpha T_d} \end{bmatrix} \begin{bmatrix} \mathbf{x_i}(t) \\ \mathbf{x_d}(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{T_i} \\ \frac{1}{\alpha^2 T_d} \end{bmatrix} \mathbf{e}(t) \end{aligned} \tag{18}$$

2.2.2 Our regulator

In our linearized model the pitch angle and jaw angle are completely separated, i.e. there are no cross couplings between these states. To control the robot we then use two separate PID control systems, (18), one to regulate pitch and another one to regulate jaw. The output signals of these regulators are then added to form the final control signal. Also since the state are not measured directly we use a kalman filter for state estimation. A schematic picture of the observer based PID control system is shown in Figure 3.

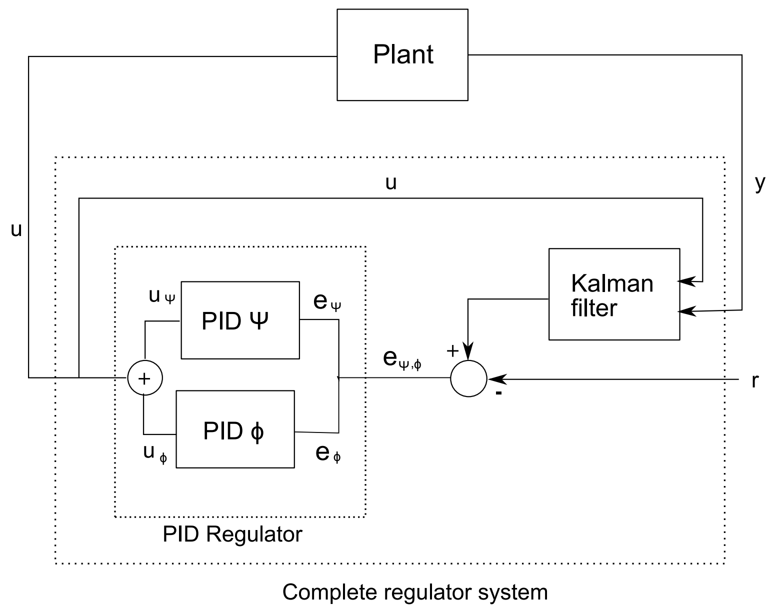


Figure 3: Figure shows a schematic picture of the complete PID controller and its internal connections. PID ψ and PID ϕ is the controllers which controls pitch and jaw respectively and r is the reference signal for ψ and ϕ.

2.2.3 Tuning of the regulator

There are several ways to tune these regulators, a popular tuning scheme is Ziegler-Nichols but unfortunately it only applies to stable systems. There are also more systematic ways to tune PID regulators, such as lead-lag compensation. Since we considered the PID only as a side project we didn't want to put effort into this controller and the tuning were done by "feeling"

2.2.4 Simulation results

In order to evaluate the regulator we have performed simulations where we control the discrete linear robot model using the control method described above. The simulation experiments test the performance in terms of balancing, turning and the linear running of the robot, which are considered to be fundamental motions of a two wheeled inverted pendulum robot. In the following section these results are presented for the PID regulator. The plots show the states in consideration, the estimates of these states and the reference values. The control signal is also shown as the last plot in each graph. The control signal in the real system have a maximum amplitude of 8 V, as long as it is kept under this value it is not saturated.

Balancing Control The balancing performance is evaluated by giving the robot a pitch velocity of $40\,deg/s$, while standing in an upright position. Figure 4 shows the results. Here we can see that the robot returns to its upright position after about 2.5 seconds.

Jaw control The turning performance is evaluated for the regulator by giving the robot a step reference for the jaw angle, while standing still at an upright position. Figure 5 shows the simulation results. We can see that the robot overshoots about 5 degrees and reaches the reference after about 2 seconds. It should be noted that the pitch angle is at zero degrees during the response. This is because in the linear model there are no cross couplings between the jaw angle and the pitch angle. We therefore expect the robot to perform slightly different than this simulation in reality.

Pseudo linear running control The controller has no explicit linear running control. This linear running motion can however be performed by ramping up the pitch reference and bringing it back down to zero. When this is done the controller accelerates the robot in the same direction as the pitch reference in order to balance the robot. When the reference is brought back to zero the controller has to keep moving the robot in the same direction with a constant speed in order to overcome the inertia of the robot. The performance is evaluated by, as described, giving the robot a ramp up and ramp down reference for the pitch angle, while standing still at an upright position. Figure 6 shows the simulation results. Here we can see that the robot reaches a velocity of 0.2 m/s and is back at it's upright position after about 2 seconds.

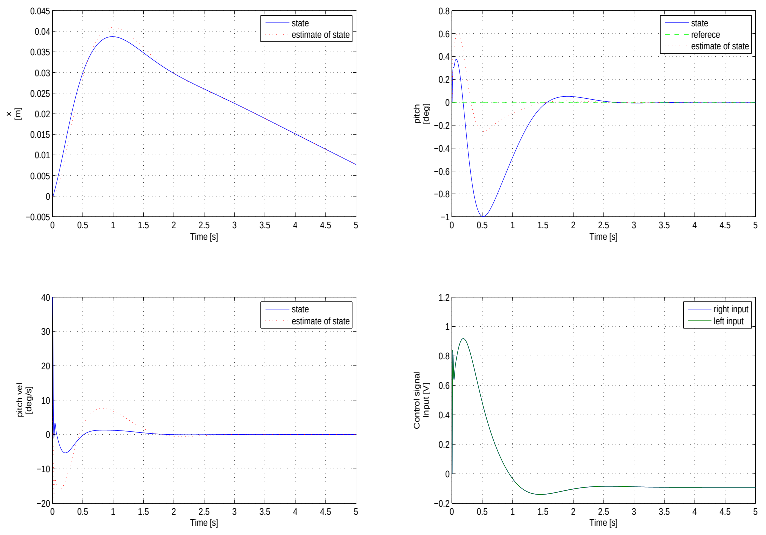


Figure 4: The x position, pitch, pitch velocity and control signal is shown when the robot is given a pitch velocity of 40 deg/sec at time 0.

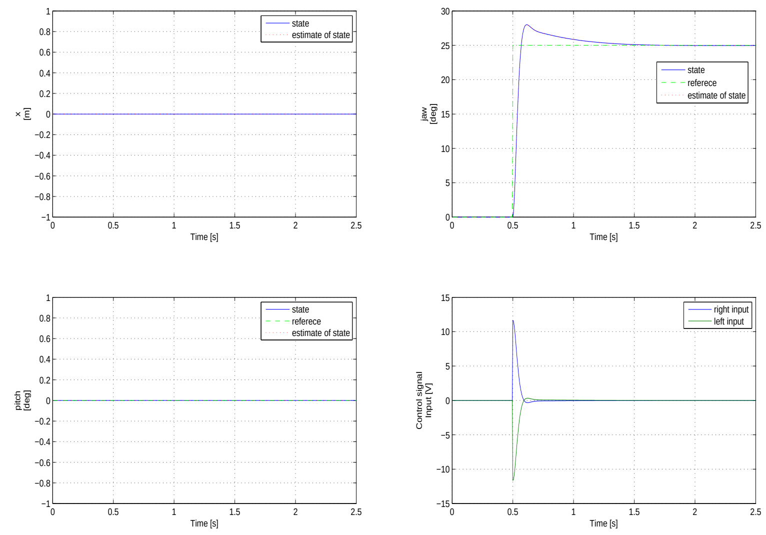


Figure 5: x, jaw, pitch and control signal is shown when a step of 25 deg in jaw reference is set at time 2.5 s.

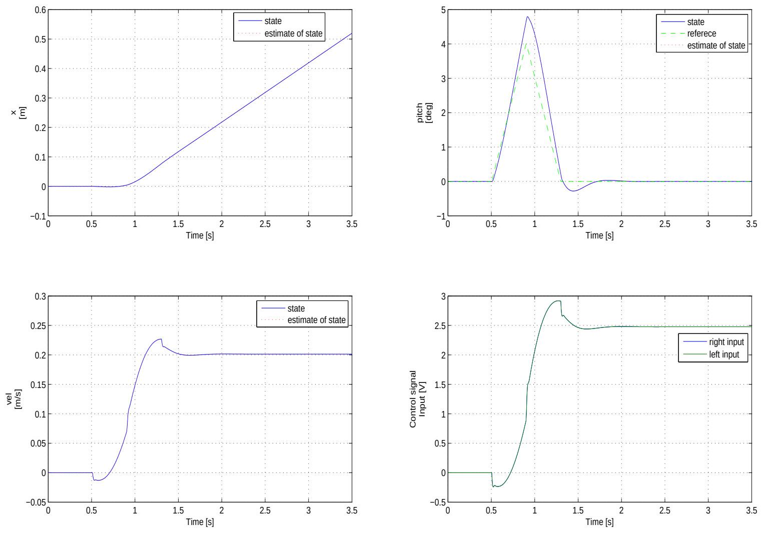


Figure 6: The x position, pitch, velocity and control signal is shown when a ramp up and ramp down reference is applied at time 0.5 - 1.25 s.

2.3 LQG Regulator with integral action

2.3.1 Short about LQG regulators

LQG regulators could be designed for linear models on state space form. If the system is linear and the disturbances are white Gaussian noise the LQG regulator will be the optimal regulator in the sense that it is the regulator that minimizes the criteria

$$J = \sum_{n} \mathbf{z}^{\mathbf{T}}(\mathrm{n})\mathbf{Q_1}\mathbf{z}(\mathrm{n}) + \mathbf{u}^{\mathbf{T}}(\mathrm{n})\mathbf{Q_2}\mathbf{u}(\mathrm{n}) \tag{19}$$

Where $\mathbf{z}$ is the performance signal, $\mathbf{u}$ is the input signal and $\mathbf{Q_1}$and $\mathbf{Q_2}$ are symmetric weighting matrices, which in most cases are diagonal. The solution to this problem is found by solving the associated Ricatti equation. From this a state feedback matrix $\mathbf{L}$ is obtained which will serve as regulator.

If not all states could be measured a Kalman filter could be used as an observer. It could be shown that the design of the Kalman filter and the gain matrix $\mathbf{L}$ are independent of each other. This greatly simplifies the design procedure.

LQG regulators will often provide good regulation even if the situation is not ideal. The model could be slightly nonlinear and the disturbances could deviate somewhat from white Gaussian noise. It is also quite easy to tune these regulators by adjusting the weighting matrices $\mathbf{Q_1}$ and $\mathbf{Q_2}$ appropriately in an iterative process. This is two of the reasons why we have chosen to use this regulator.

2.3.2 Design of LQG regulator

Making the regulator a servo controller The robot should be able to follow a reference signal, $r(n)$, we must therefore make a servo controller for it. If $\mathbf{e}(n) = \mathbf{z}(n) - \mathbf{r}(n)$ is the control error, which is the quantity that we would like to keep small, expression (19) could be appropriately modified to create a state feedback for this situation. However it could be shown that if the reference signal is assumed to be piecewise constant the same feedback gain matrix $\mathbf{L}$ as if $\mathbf{e}(n) = \mathbf{z}(n)$ (i.e. $\mathbf{r}(n) = 0$) would be obtained, this is the approach we will adapt. To get unit gain from the reference signal to the performance signal in stationarity, a gain matrix $\mathbf{L_r}$ is placed in the feed forward path. Thus the control law with observer based state feedback structure is,

$$\mathbf{u}(n) = -\mathbf{L}\hat{\mathbf{x}}(n) + \mathbf{L_r}\mathbf{r}(n) \tag{20}$$

Introducing integral action in the regulator LQG regulators don't provide inherent integral action. Since we desire integral action in the control loop in order to get rid of static errors we add a term $\mathbf{e_i^T Q_i e_i}$ to the criteria (19), which penalizes the integrated control error, $\mathbf{e_i}(n)$. On state space form the integrating system would be

$$\mathbf{e_i}(n+1) = (1-\delta)\mathbf{e_i}(n) + T_s\overbrace{[\mathbf{Mx}(n) - \mathbf{r(n)}]}^{e(n)} \tag{21}$$

Where δ is some small scalar which will take the system poles into the stability region. To be able to work within the LQG framework $\mathbf{e_i}(n)$ is introduced to the system as a fictitious state. Thus the integrating system (21) is augmented with the original system (15). The augmented system then becomes

$$\begin{bmatrix} \mathbf{x}(n+1) \\ \mathbf{e_i}(n+1) \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{T_s M} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x}(n) \\ \mathbf{e_i}(n) \end{bmatrix} + \begin{bmatrix} \mathbf{B} \\ \mathbf{0} \end{bmatrix} \mathbf{u}(n) + \begin{bmatrix} \mathbf{H} & \mathbf{0} \\ \mathbf{0} & -\mathbf{T_s I} \end{bmatrix} \begin{bmatrix} \mathbf{w}(n) \\ \mathbf{r}(n) \end{bmatrix} \tag{22}$$

$$\begin{bmatrix} \mathbf{z}(n) \\ \mathbf{e_i}(n) \end{bmatrix} = \begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x}(n) \\ \mathbf{e_i}(n) \end{bmatrix} \tag{23}$$

$$\mathbf{y}(n) = \begin{bmatrix} \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}(n) \\ \mathbf{e_i}(n) \end{bmatrix} + \mathbf{v}(n) \tag{24}$$

The regulator is then designed for this system. By this "trick" we will achieve integral action in the regulator. Some important observations can be made here. In the system above $\mathbf{r}(n)$ is modeled as a disturbance, this is valid since from the controllers point of view $\mathbf{r}(n)$ can be regarded as just that, a disturbance. Also since the extra states, in $\mathbf{e_i(n)}$, are fictitious, and present only in the control loop, the augmented system is only considered when designing the state feedback gain matrix **L.** The Kalman filter design is still based on the original system.

2.3.3 Schematic picture of the regulated system

Figure 7 shows a schematic picture of the complete regulator operating in closed loop.

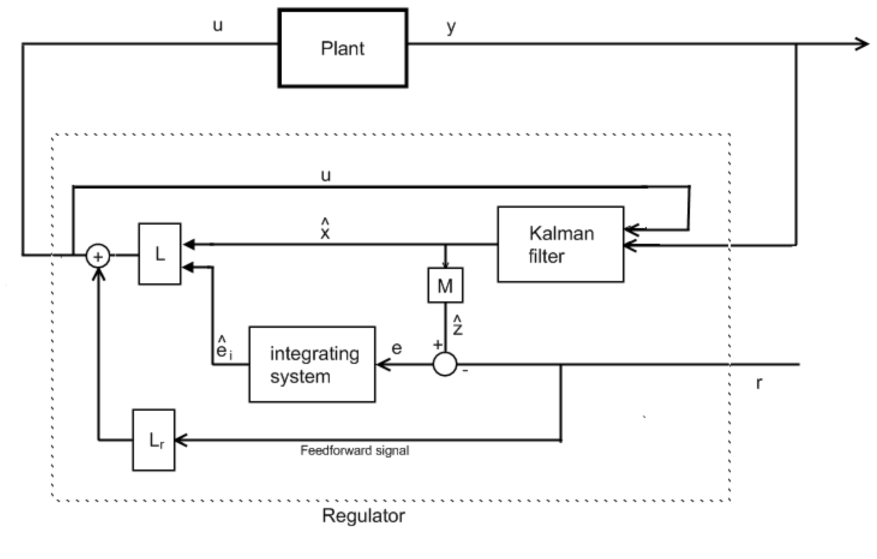


Figure 7: A schematic picture of the closed loop system, the regulator takes as input the reference signal r and the measured signal y and outputs the corresponding control signal u.

2.3.4 Our regulator

Our regulator is constructed as shown in Figure 7. The state-space representation of the control system is given by,

$$\begin{aligned} \mathbf{u}(n) &= -\begin{bmatrix} \mathbf{L_x} & \mathbf{L_i} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}}(n) \\ \mathbf{e_i}(n) \end{bmatrix} + \begin{bmatrix} \mathbf{L_r} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{r}(n) \\ \mathbf{y}(n) \end{bmatrix} \\ \begin{bmatrix} \hat{\mathbf{x}}(n+1) \\ \mathbf{e_i}(n+1) \end{bmatrix} &= \begin{bmatrix} \mathbf{A} - \mathbf{KC} - \mathbf{BL_x} & -\mathbf{BL_i} \\ \mathbf{T_s M} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{x}}(n) \\ \mathbf{e_i}(n) \end{bmatrix} + \begin{bmatrix} \mathbf{BL_r} & \mathbf{K} \\ -\mathbf{T_s I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{r}(n) \\ \mathbf{y}(n) \end{bmatrix} \end{aligned} \tag{25}$$

Here $\mathbf{L} = \begin{bmatrix} \mathbf{L_x} & L_i \end{bmatrix}$.

The regulator will as performance signal use jaw angle, pitch angle and forward velocity. The $\mathbf{M}$ matrix in (15) is then given by

$$\mathbf{M} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Tuning of the regulator The regulators were tuned by adjusting the weighting matrices so that simulations showed good results in regulation of disturbances and ability to follow reference signals without saturating the input signal, see section 2.3.5 for simulation results. The regulator were then tested on the real system. Since there are imperfections in the physical model the results here were as expected not exactly the same. By looking at the behavior of the real system we then adjusted the weighting matrices so that we achieved good regulation.

2.3.5 Simulation results

In order to evaluate the regulator we have performed simulations where we control the discreet linear robot model using the control method described above. The simulation experiments test the performance in terms of balancing, linear running and turning of the robot, which are considered to be fundamental motions of a two wheeled inverted pendulum robot. In the following section these are presented for the regulator. The plots shows the states in consideration, the estimates of these states and the reference values for each state. The control signal is also shown as the last plot in each graph. The control signal in the real system have a maximum amplitude of about 8 V, as long as it is kept under this value it is not saturated.

Balancing Control The balancing performance is evaluated for both regulators by giving the robot a pitch velocity of 40 degrees/s, while standing in an upright position. Figures 8 and **??** show the results for the moving and still regulator. For both controllers the robot returns to its upright position after about 5 seconds.

Linear running control Linear running control is only implemented in the "moving regulator" and thus only simulated for this controller. The performance is evaluated by giving the robot a step reference for the linear velocity, while standing still at an upright position. Figure 9 shows the simulation results. Here we can see that the robot reaches the velocity reference and is back at its upright position after about 2 seconds.

Jaw control The turning performance is evaluated for both regulators by giving the robot a step reference for the jaw angle, while standing still at an upright position. Figure 9 shows the simulation results. We can see that the for the "moving regulator" the robot overshoots about 5 degrees and reaches the reference after about 2 seconds while the "still regulator" has no over shoot and reaches the reference with a small deviation after about 1.5 seconds. This is because the "move controller" is implemented to respond harder on the jaw reference while the "still regulator" designed to give a softer response. It should be noted here that the pitch angle is at zero degrees during the response. This is because in the linear model there are no cross couplings between the jaw angle and the pitch angle. We therefore expect the robot to perform slightly different then this simulation.

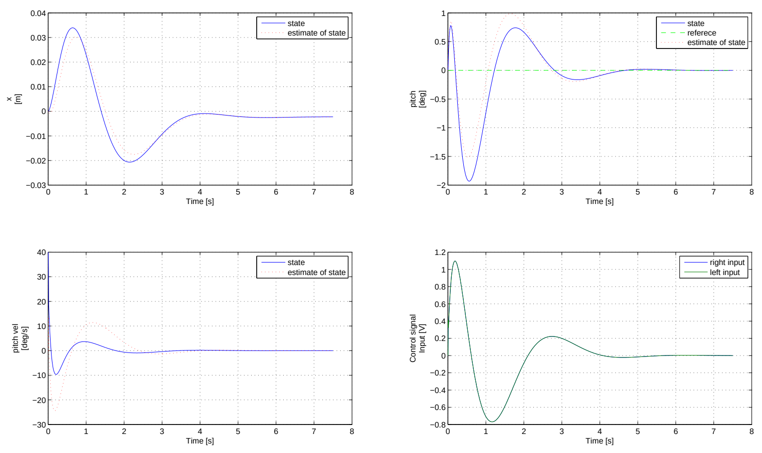


Figure 8: The x position, pitch, pitch angular velocity and control signal is shown when the robot is given a pitch velocity of $40\,deg/sec$ at time 0.

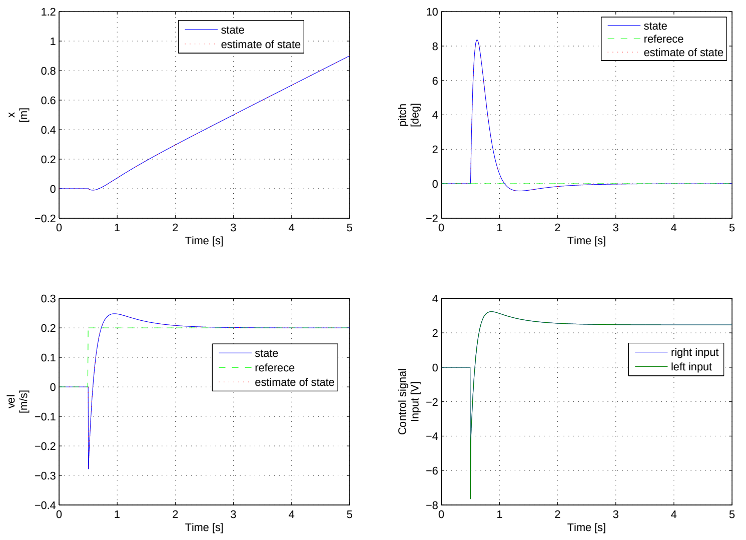


Figure 9: The x position, pitch, velocity and control signal is shown when a step of 0.2 m/s in velocity reference is applied at time 0.5 s.

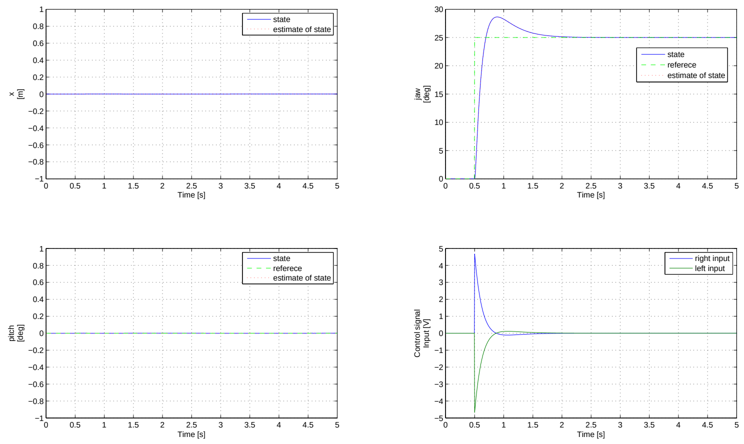


Figure 10: x, jaw, pitch and control signal is shown when a step of 25 deg in jaw reference is set at time 2.5 s.

Simulation results LQG Regulator

3 Line tracker

3.1 Introduction

It is desired that the robot is able to follow a dark line. To sense the line the robot uses a light sensor. Based on the reflected light intensity the robot is able to distinguish between a dark area ("on the track") and a light area ("off the track"). This is done by comparing the light sensor value with a threshold value, with some hysterisis. If the value is less than the threshold the robot is assumed to be on the track, otherwise it is assumed to be off the track. That information is then used by the line tracking algorithm to stay on the track.

There is no real restriction on how the track could be shaped, as long as the track is continous. So the robot software should not make use of any a priori information about the track shape in its line-tracking algorithm.

An example on how a track could look like is shown in Figure 11. This track also available as a print-out. Unfortunately the line is printed using a rather light green color. This will make it harder to distinguize between "on track" and "off track"

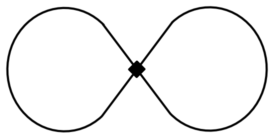
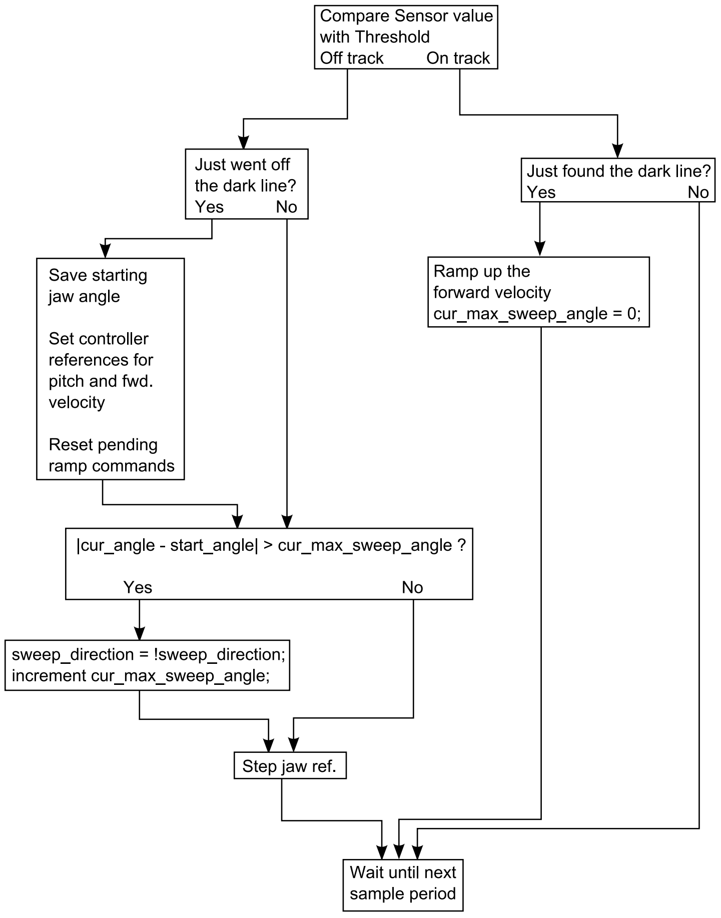


Figure 11: The Eight ShapedTrack

3.2 The algorithm

The idea behind the line-tracking algorithm is to stop and sweep the area in front of the robot when the robot is not on the track. The sweeping is performed by first turning the jaw a maximum of eg. 10° in one direction. When it is done the robot will return back to the initial jaw angle, i.e. the angle where it lost the track. The robot will then try to sweep the area in the opposite direction. This time the maximum sweep angle has been extended by eg. 5°. The procedure is then repeated until the line is located. So the sweeping area will keep growing at every direction change. As soon has the line has been located the robot stops the sweeping and starts to move forward. The algorithm remembers in which direction the track was found last time. So the next time the robot comes off the track the algorithm starts sweeping in that direction.

A detailed flow-chart on how the line-tracker algorithm works is depicted in Figure 12. One loop is performed every sample period. For a more illustrative example see Figure 13.



Compare Sensor value with Threshold
Off track
On track
Just went off the dark line?
Yes
No
Just found the dark line?
Yes
No
Save starting jaw angle
Set controller references for pitch and fwd. velocity
Reset pending ramp commands
Ramp up the forward velocity
cur_max_sweep_angle = 0;
|cur_angle - start_angle| > cur_max_sweep_angle ?
Yes
No
sweep_direction = !sweep_direction;
increment cur_max_sweep_angle;
Step jaw ref.
Wait until next sample period

Figure 12: Line-tracker algorithm

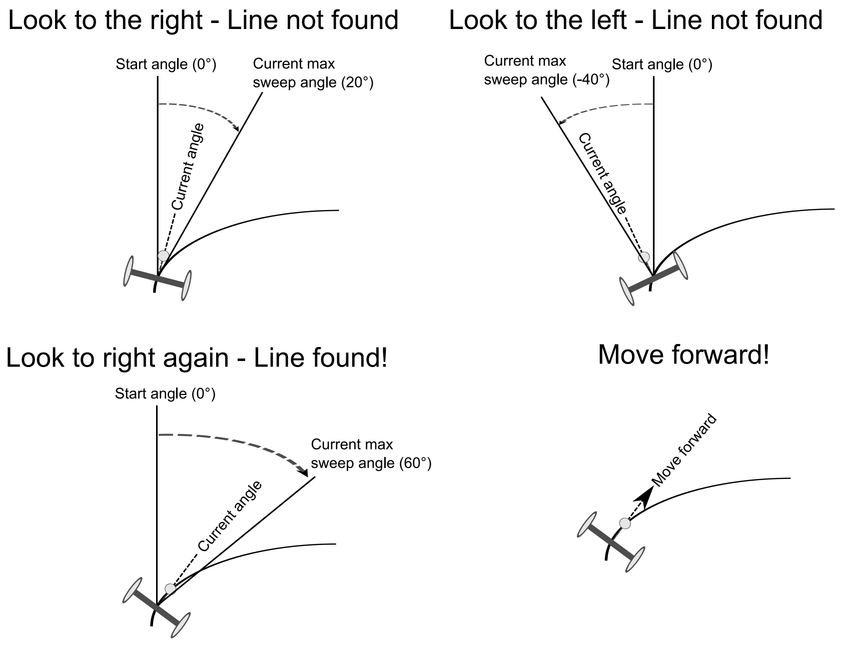

Look to the right - Line not found
Start angle (0°)
Current max sweep angle (20°)
Current angle
Look to the left - Line not found
Current max sweep angle (-40°)
Start angle (0°)
Current angle
Look to right again - Line found!
Start angle (0°)
Current max sweep angle (60°)
Current angle
Move forward!
Move forward

Figure 13: Example run of the line-tracking algorithm

4 Software

4.1 Lego software

Once a regulator has been developed on the PC it is finally time to test the regulator on the real robot. The task for the Lego software is mainly to try to balance the robot using that regulator. Apart from that the software has some more features built-in.

4.1.1 Functionality

A feature which comes in handy when evaluating the performance of a controller is the data logger. With the logger it is possible to log the variables holding information about the input to the controller (references and sensor values), the output from the controller (the two motor voltages) and the internal states of the controller to a PC (via USB or Bluetooth). To save on the available data log memory it is possible to choose exactly which regulator inputs, outputs and states that should be logged. This feature is almost necessary since the data log in the robot has only room for about 1500 data points. The logs can then be transferred to a PC for evaluation in a Matlab script. See Appendix for more detailed information.

The robot can take input from a gamepad connected to a PC. The buttons control the robot by applying reference signals to the controller. The references are applied gradually by ramping up them to some final value in smaller steps.

One of the gamepad buttons is used to activate/deactivate the line tracker. The robot is then able to follow a dark line.

The robot can be stopped and then restarted by the press of another gamepad button.

4.1.2 Implementation

The program is implemented using only one task. The reason for this is to avoid the complexity that comes with multitasking systems (overhead, synchronization, deadlines etc.).

Once the program has initialized everything it needs it will enter the big control loop. It is in here where everything happens. See Figure 14 for a full flow-chart of the program.

Since the controller, which is supposed to balance the robot, is designed for a certain sampling frequency it is important that all timings get correct. For example the control signal should be applied at the same time instant as the sensors have been read and the new control signal has been calculated. That is the reason why the more computational heavy parts are postponed until after the control signal has been applied. It is also vital that the control loop does not exceed the sample period. For that reason a control mechanism is implemented which will simply stop the robot if the sample period is overdued.

Unfortunately, for the gamepad to work the sample period sometimes might be exceeded by 1-3 ms. To get as little influence on the balancing as possible the gamepad is only read once every 200 ms. With such low read frequency the balancing should not be affected noticeably.

The controllers are represented in a linear state-space form. Calculating the control signal and iterating the internal states of the controller therefore involve performing matrix-vector multiplications. Those operations become quite demanding for the Lego system as the state-space matrices become large. An alternative method to writing general matrix-vector multiplication routines (typically involving for-loops) is to generate a set of sums of scalar multiplications in RobotC code. The latter method is used in this program. All matrix-vector multiplications that are to be performed by the robot are generated by a Matlab script. Given a controller the script generates the RobotC code files necessary for the Lego software to operate the controller. This method also gives the script the opportunity to simply skip generating multiplications for matrix elements which are zero.

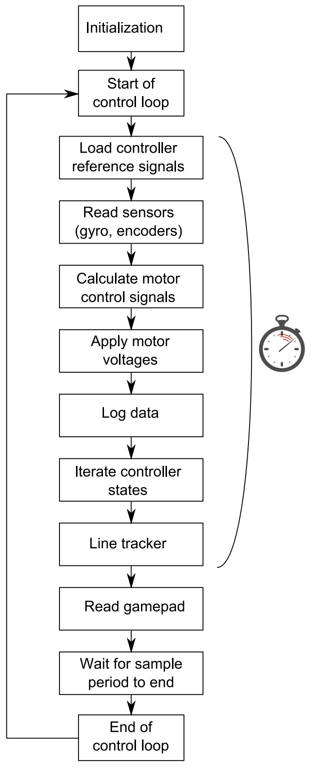

Initialization
Start of control loop
Load controller reference signals
Read sensors (gyro, encoders)
Calculate motor control signals
Apply motor voltages
Log data
Iterate controller states
Line tracker
Read gamepad
Wait for sample period to end
End of control loop

Figure 14: Robot software flowchart

5 Appendix

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & a_{43} & a_{44} & 0 & a_{46} \\ 0 & 0 & 0 & 0 & a_{55} & 0 \\ 0 & 0 & a_{64} & a_{64} & 0 & a_{66} \end{bmatrix}$$

$$den = \left(m_b\, L^2 + I_p + 2J_m\right) \left(m_b + 2(m_w + \frac{I_w + J_m}{R^2})\right) - \left(m_b L - \frac{2\, J_m}{R}\right)^2$$

$$a_{43} = -\frac{\left(m_b L - \frac{2\, J_m}{R}\right) m_b\, L\, g}{den}$$

$$a_{44} = -\frac{\frac{2}{R^2}\left(B_m + \frac{K_b\, K_t}{R_a}\right) \left((R+L) m_b L + \mathrm{I_p}\right)}{den}$$

$$a_{46} = \frac{\frac{2}{R}\left(B_m + \frac{K_b\, K_t}{R_a}\right) \left((R+L) m_b L + \mathrm{I_p}\right)}{den}$$

$$a_{55} = -\frac{2\, W^2 \left(B_m + \frac{K_b\, K_t}{R_a}\right)}{R^2\, (I_j + 2\, W^2\, m_w) + 2\, W^2\, (I_w + J_m)}$$

$$a_{63} = \frac{\left(m_b + 2(m_w + \frac{I_w + J_m}{R^2})\right) m_b\, L\, g}{den}$$

$$a_{64} = \frac{\frac{2}{R^2}\left(B_m + \frac{K_b\, K_t}{R_a}\right) \left((R+L) m_b + 2(\mathrm{m_w} + \frac{\mathrm{I_w}}{\mathrm{R^2}})\mathrm{R}\right)}{den}$$

$$a_{66} = -\frac{\frac{2}{R}\left(B_m + \frac{K_b\, K_t}{R_a}\right) \left((R+L) m_b + 2(\mathrm{m_w} + \frac{\mathrm{I_w}}{\mathrm{R^2}})\mathrm{R}\right)}{den}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ b_1 & b_2 \\ b_3 & b_4 \\ b_5 & b_6 \end{bmatrix}$$

$$b_1 = \frac{\frac{K_t}{R\,R_a}\left((R+L)\,m_b L + I_p\right)}{den}, \qquad b_2 = b_1$$

$$b_3 = \frac{\frac{K_t}{R\,R_a} W}{(I_J + 2(m_w + \frac{I_w + J_m}{R^2})W^2)}, \qquad b_4 = -b_3$$

$$b_5 = -\frac{\frac{K_t}{R\,R_a}\left((R+L)\,m_b + 2(m_w + \frac{I_w}{R^2})R\right)}{den}, \qquad b_6 = b_5$$

$$\mathbf{C} = \begin{bmatrix} \frac{1}{R} & \frac{W}{R} & -1 & 0 & 0 & 0 \\ \frac{1}{R} & -\frac{W}{R} & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

5.1 Overview of the programming design process

A schematic picture of how we construct and test regulators is shown in Figure 15.

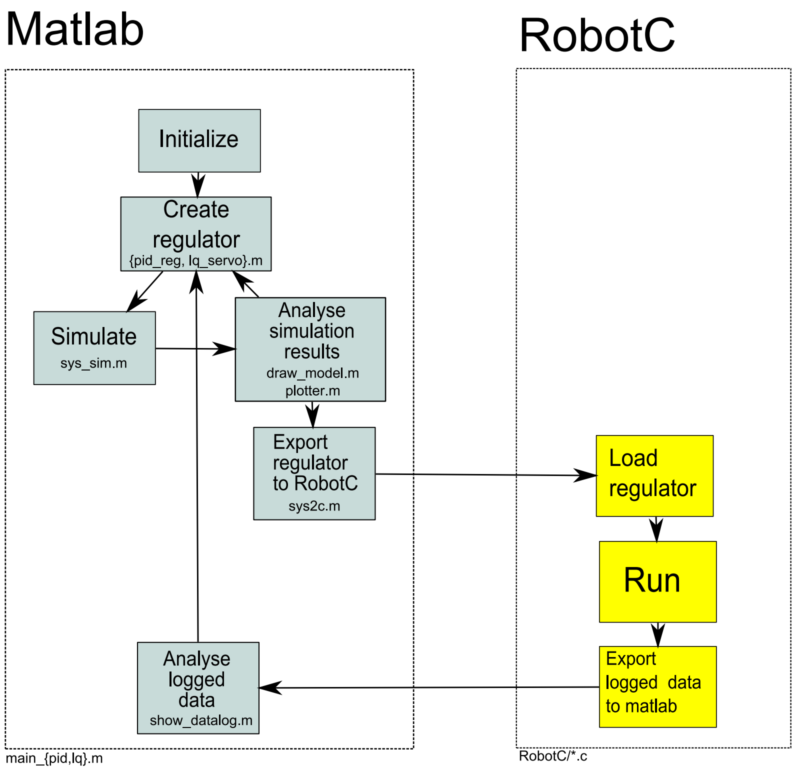


Figure 15: A schematic picture showing the process of creating a regulator and testing it on the real system.

A more detailed explanation of each block in Figure 15 is given below.

5.1.1 In Matlab

Initialization

Constants and state space model are loaded from constants.m and statespace_model.m respectively.

Create regulator

The regulator is constructed by specifying the needed regulator parameters and then calling lq_servo.m/pid_reg.m with these parameters as input variables. These functions will give as output a complete regulator system on state space form, including the Kalman system.

Simulate

Simulations of the linearized system is done by running sys_sim.m which will return the simulation results.

Analyse simulation results

The obtained results from execution of sys_sim.m are presented by using plotter.m which will display simulation plots. draw_model.m will show a simple animation of the simulation.

Export regulator to RobotC

If the simulation results look satisfactory the regulator is exported to RobotC by calling sys2c.m.

5.1.2 In RobotC

Run the software on the Lego device

To build and upload the RobotC program to the robot start “RobotC for Mindstorms” and press F5. Make sure that the robot is not moving. Then press F5 once more to start the code execution. The robot will then make a short beep. When the gyro bias has been calculated the robot will play a short melody. This means that you shall place the robot in the upright balancing position. After the third and last beep the robot is ready to balance!

Control the robot from a gamepad

Once the robot is balancing the robot can be controlled by a gamepad which is connected to a PC. Make sure that the RobotC gamepad window is opened from the "Robot -> Debug -> Joystick" menu. The following functions can then be called by pressing the corresponding gamepad button.

Command	Button
Inc. velocity	4
Dec. velocity	1
Turn left	3
Turn right	2
Active/deactive line-tracker	6
Stop/start	7
Calibrate line-tracker	8
Horn	5

Activate data logging

To activate the data logger choose the file "log_settings.c" from the "Include" menu. Then set the variables that you wish to log to the digit "1" and the those that you do not want to log to the digit "0". Then active the logging by setting LOG_DATA to "1".

Analyze logged data

The logged data first has to be transfered to the PC. This is done from the "NXT Brick -> File management" menu. The data log is named "DATAxxxx". Choose "Upload spreadsheet" to download the file to the PC. Change to the directory containing all the RobotC source files. Then open the file "show_data log.m".

Specify the filename of the saved data log. Then run that "cell" by pressing [Ctrl]+[Enter]. This will load the filename and the regulator to the workspace.

In the next cell change "log_period" to reflect how often the data was saved, typically every sample period (lego_reg.Ts). The "NUM_DEC" tells how many decimals were used when the data was logged. See "DECIMAL_FAC" in the RobotC source file "log_settings.c" for more information. Then run the cell ([Ctrl]+[Enter]) to load the saved data to workspace.

Finally choose which variables that you wish to plot and run that cell ([Ctrl]+[Enter]).

References

[1] Yamamoto Y, (2008) NXTway-GS Model-Based Design - Control of self-balancing two-wheeled robot built with LEGO Mindstorms NXT

[2] Jeong S, Takahshi T (2007) Wheeled inverted pendulum type assistant robot: Design concept and mobile control.

[3] Grasser F, D'arigo A, Colombi S, Rufer A (2001) JOE: A mobile, inverted pedulum

[4] http://www.hitechnic.com/cgi-bin/commerce.cgi?preadd=action&key=NGY1044

[5] http://mindstorms.lego.com/Products/Sensors/Light%20Sensor.aspx

[6] http://mindstorms.lego.com/Products/Accessories/Interactive%20Servo%20Motor.aspx

[7] http://www.philohome.com/nxtmotor/nxtmotor.htm

[8] http://web.mac.com/ryo_watanabe/iWeb/Ryo%27s%20Holiday/NXT%20Motor.html

[9] Glad T, Ljung L (2006) Reglerteknik grundläggande teori

[10] Glad T, Ljung L (2003) Reglerteori flervariabla och olinjära metoder