Design of the Dragon control system

Jan Vangorp

November 17, 2009

Contents

1 Model of the Dragon

1.1 Mechanical model

A sketch of the mechanical model of the Dragon is drawn in figure 1. The basic structure consists of 2 wheels, each connected to a motor, the body of the Dragon and on top the battery.

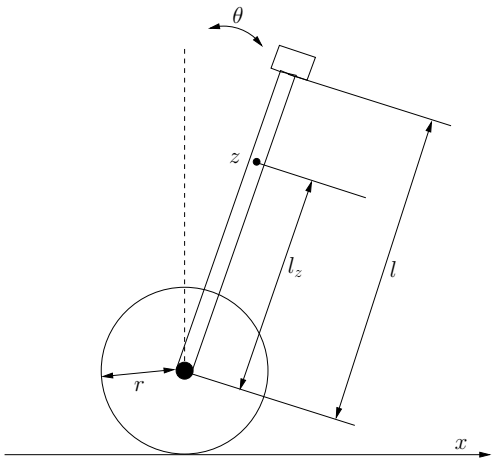

Figure 1: Dragon: mechanical sketch

We will model the wheels as discs and the body as a rod. The battery and motors are each modeled as point masses. This results in the following mechanical parameters of the Dragon:

length of the rod	l	0.56 m
radius of the wheels	r	0.05 m
mass of a wheel	m_w	0.5 kg
mass of the battery	m_b	1.874 kg
mass of the motors	m_m	1.8 kg
mass of the rod	m_r	2.3 kg
total mass of the motors, rod and battery	m_z	5.974 kg

Center of gravity of the motors+rod+battery:

$$l_z = \frac{m_b l + m_r l/2}{m_b + m_r + m_m} \text{ [m]} \tag{1}$$

Moment of inertia of wheel around axle:

$$I_w = \frac{m_w r^2}{2} \text{ [kg.m}^2\text{]} \tag{2}$$

Moment of inertia of body+battery around axle:

$$I_b = \frac{m_r l^2}{3} + m_b l^2 \text{ [kg.m}^2\text{]} \tag{3}$$

Moment of inertia of motors+body+battery around center of gravity:

$$I_z = \frac{(l_z/l)m_r l_z^2}{3} + \frac{((l-l_z)/l)m_r(l-l_z)^2}{3} + m_b(l-l_z)^2 + m_m l_z^2 \text{ [kg.m}^2\text{]} \tag{4}$$

To derive the motion equations of the Dragon, we will treat it as two parts: the wheels and the motors+body+battery. For both parts the mechanical equilibrium is written down.

Starting with the wheel, the free body diagram is shown in figure 2(a).

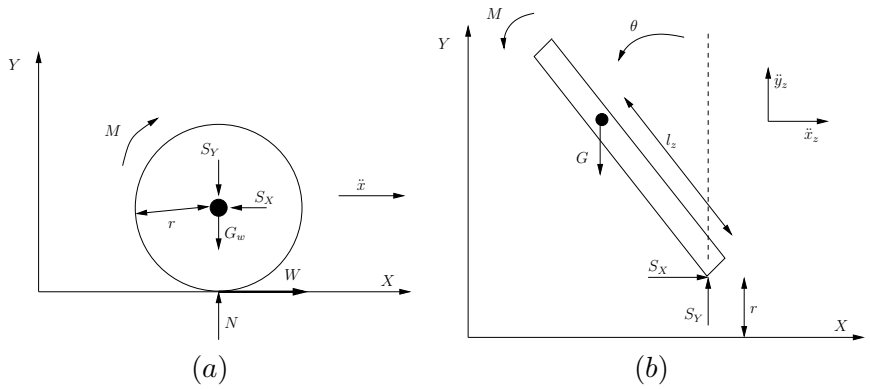

Figure 2: Free body diagrams: (a) wheel, (b) motors+body+battery

The equilibrium $ma = \sum F$ of forces in the horizontal direction and the equilibrium of moments $I\alpha = \sum M$ lead to respectively

$$m_w\ddot{x} = W - S_X \tag{5}$$
$$I_w\alpha = -M + rW \tag{6}$$

Note that the equilibrium of forces in the vertical direction does not give us any information since we assume that the Dragon does not bounce. From these equations we can eliminate the friction force W and using $\alpha = -\frac{\ddot{x}}{r}$ the equation for the wheel becomes

$$\left(-\frac{I_w}{r^2} - m_w\right)\ddot{x} = -\frac{M}{r} + S_X \tag{7}$$

where M is the moment by the motor and S_X is a force from the body of the Dragon. Note that when there are two wheels, each wheel will experience only $S_X/2$.

For the motors+body+battery, the free body diagram is shown in figure 2(b). Again we can write down the equilibrium of forces (in both directions this time: the body does move vertical!) and moments (we can calculate the moment around any point, we chose the center of gravity):

$$m_z\ddot{x}_z = S_X \tag{8}$$
$$m_z\ddot{y}_z = S_Y - G \tag{9}$$
$$I_z\ddot{\theta} = M + S_X l_z\cos(\theta) + S_Y l_z\sin(\theta) \tag{10}$$

Substituting the equations of the forces into the equation of the moments we get

$$I_z\ddot{\theta} = M + m_z\ddot{x}_z l_z \cos(\theta) + (m_z\ddot{y}_z + G) l_z \sin(\theta) \tag{11}$$

The position of the center of gravity $\{x_z, y_z\}$ is governed by the following equations:

$$\begin{aligned}
x_z &= -l_z \sin(\theta) + x && (12)\\
\dot{x}_z &= -l_z \cos(\theta)\dot{\theta} + \dot{x} && (13)\\
\ddot{x}_z &= l_z \sin(\theta)\dot{\theta}^2 - l_z \cos(\theta)\ddot{\theta} + \ddot{x} && (14)\\
y_z &= l_z \cos(\theta) + r && (15)\\
\dot{y}_z &= -l_z \sin(\theta)\dot{\theta} && (16)\\
\ddot{y}_z &= -l_z \cos(\theta)\dot{\theta}^2 - l_z \sin(\theta)\ddot{\theta} && (17)
\end{aligned}$$

Substituting the accelerations into (11) leads to the following equation for the motors+body+battery:

$$I_z\ddot{\theta} = M - m_z l_z^2\ddot{\theta} + m_z l_z \cos(\theta)\ddot{x} + m_z g l_z \sin(\theta) \tag{18}$$

With (8) and (14) we now also have an expression for S_X to complete equation (7) of the wheel:

$$\left(-\frac{I_w}{r^2} - m_w\right)\ddot{x} = -\frac{M}{r} + m_z l_z \sin(\theta)\dot{\theta}^2 - m_z l_z \cos(\theta)\ddot{\theta} + m_z\ddot{x} \tag{19}$$

The motion equations of the (one wheeled) Dragon are thus

$$\begin{aligned}
-m_z l_z \cos(\theta)\ddot{\theta} + m_z l_z \sin(\theta)\dot{\theta}^2 + (m_z + I_w/r^2 + m_w)\ddot{x} &= \frac{M}{r} && (20)\\
(I_z + m_z l_z^2)\ddot{\theta} - m_z l_z \cos(\theta)\ddot{x} - m_z l_z g \sin(\theta) &= M && (21)
\end{aligned}$$

In the appendix, an alternative, more concise but more abstract derivation of these equations is outlined.

Clearly, the motion of the Dragon is governed by nonlinear equations. However, we would like to apply linear systems theory to the Dragon in order to derive a control structure. Noting that in the end we want to keep the Dragon upright with only small deviations, we can linearize the motion equations around $\theta = 0$ using $\sin(\theta) \approx \theta \approx 0$ and $\cos(\theta) \approx 1$:

$$\begin{aligned}
-m_z l_z\ddot{\theta} + (m_z + I_w/r^2 + m_w)\ddot{x} &= \frac{M}{r} && (22)\\
(I_z + m_z l_z^2)\ddot{\theta} - m_z l_z g\theta - m_z l_z\ddot{x} &= M && (23)
\end{aligned}$$

Or in the more familiar Laplace domain:

$$\begin{aligned}
-m_z l_z s^2\theta + (m_z + I_w/r^2 + m_w)s^2 x &= \frac{M}{r} && (24)\\
(I_z + m_z l_z^2)s^2\theta - m_z l_z g\theta - m_z l_z s^2 x &= M && (25)
\end{aligned}$$

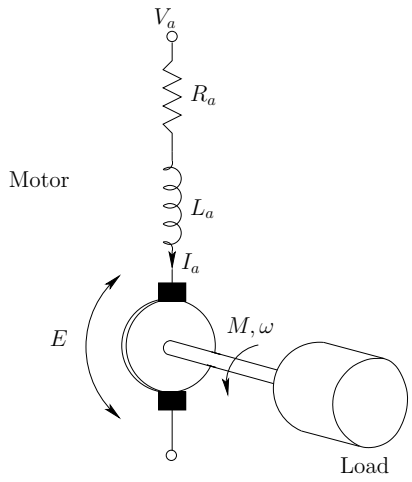

Figure 3: Dragon: electrical sketch

1.2 Electromechanical model

The Dragon is driven by 2 DC motors attached to the wheels. A sketch of a DC motor is given in figure 3. By using a DC motor, we can easily control the speed: the back electromotive-force voltage E is proportional to the speed of the rotor:

$$E = K_e \omega \tag{26}$$

When applying a voltage V_a to the motor, a current I_a flows that depends on the back electromotive-force voltage:

$$V_a - E = I_a(R_a + sL_a) \tag{27}$$

where R_a and L_a are the parasitary resistance and inductance of the windings. In turn, the current through the motor will allow the motor to generate a torque on its axle. This moment M is proportional to the current:

$$M = K_m I_a \tag{28}$$

In steady state condition, the electrical power input to the motor is equal to the mechanical power delivered to the load: $EI_a = M\omega$. It follows that both proportionality constants are equal: $K_e = K_m = K$, the motor constant.

When the moment is applied to the mechanical load, it will start rotating with some speed ω, which in turn results in some back electromotive-force voltage E,... and a feedback system is formed. Assuming the transfer function of the mechanical load is $\frac{\omega(s)}{M(s)}$, a block diagram of the complete electromechanical model of the Dragon is shown in figure 4.

The parameters of the motors used on the Dragon are as follows:

motor inductance	L_a	2 mH
motor resistance	R_a	5 Ω
motor constant	K	0.926 Nm/A

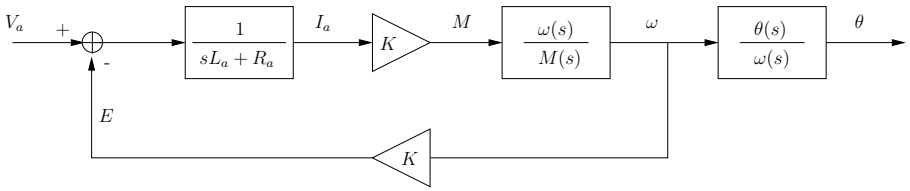

Figure 4: Dragon: electromechanical block diagram

The mechanical transfer functions $\frac{\omega(s)}{M(s)}$ and $\frac{\theta(s)}{\omega(s)}$ can be easily derived from the motion equations in the Laplace domain (24-25):

$$\frac{\omega(s)}{M(s)} = \frac{1}{r^2} \frac{(I_z + m_z l_z^2 + m_z l_z r)s^2 - m_z l_z g}{\left(-m_z^2 l_z^2 + (I_z + m_z l_z^2)(m_z + \frac{I_w}{r^2} + m_w)\right) s^3 - m_z l_z (m_z + \frac{I_w}{r^2} + m_w) g s} \tag{29}$$

$$\frac{\theta(s)}{\omega(s)} = r \times \frac{\left(m_z + \frac{2I_w}{r^2} + 2m_w + m_z l_z / r\right) s}{\left((I_z + m_z l_z^2)/r + m_z l_z\right) s^2 - m_z l_z g / r} \tag{30}$$

Again, note that the expression for $\frac{\omega(s)}{M(s)}$ is for one wheel! For a two wheeled Dragon with independent traction, this equation becomes (per wheel):

$$\frac{\omega(s)}{M(s)} = \frac{2}{r^2} \frac{(I_z + m_z l_z^2 + m_z l_z r)s^2 - m_z l_z g}{\left(-m_z^2 l_z^2 + (I_z + m_z l_z^2)(m_z + \frac{2I_w}{r^2} + 2m_w)\right) s^3 - m_z l_z (m_z + \frac{2I_w}{r^2} + 2m_w) g s} \tag{31}$$

Intuitively, one motor only has to drive half of the Dragon. So a certian torque of the motor gives you dubble the speed. Secondly, from the viewpoint of one motor, the other wheel has no traction and is just 'spinning along', resulting in an extra rotation and translation inertia.

2 Control theory

When the output of a process/device needs to be controlled, usually a feedback system is used. In this type of control system, the actual output Y is compared to the desired output D and the error E is fed into a controller C that transforms it into a steering signal S for the process/device P. A block diagram of such a control system is shown in figure 5.

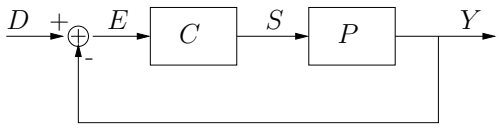

Figure 5: Basic block diagram of a feedback control system

It can be easily derived that the closed loop transfer function from D to Y is

$$\frac{Y}{D} = \frac{CP}{1 + CP} \tag{32}$$

When the transfer function of the open loop system CP provides a high gain, the transfer function of the closed loop system approaches 1 and the output will indeed follow the desired output, even when there are disturbances or deviations to the process/device.

The downside of such a feedback system is that in general stability of the closed loop system cannot be guaranteed. When designing the controller one has to explicitly check the stability. From formula (32) it can be seen that the poles of the closed loop system are different from the poles of the open loop system, resulting in a possibly completely different dynamic behaviour of the system. This allows us to make an unstable process/device stable, but it can also easily happen that a stable process/device becomes unstable!

The goal of building a feedback system around a process/device is to

- make an unstable process/device stable
- alter the dynamic behaviour of the process/device such that is falls within specifications

The dynamic behaviour is usually measured by applying a step signal. From the resulting step response, one can determine the rise time, settling time, overshoot and steady state error. These parameters show how fast and how accurately the output of the system adapts to a change in the desired output:

- *rise time*: time to go from 10% to 90% of the steady state value
- *settling time*: time required to settle within 2% of the steady state value
- *overshoot*: amount (usually in %) that the output maximally rises above the desired response
- *steady state error*: error of the output in comparison with the desired response when the transient response has decayed

An example of a step response is shown in figure 6.

2.1 PID control

Open loop performance of a PID-controller: the control configuration

In industrial environments, the PID-controller is still a favourite feedback controller: practical experience confirms that the PID-controller has a structure which is sufficiently versatile to cope with a multitude of process control problems. Other advantages of the PID-law are the low number of design parameters and the fact that the controller parameters can be easily related to performance measures.

The PID-controller shown in figure 7 is a linear controller. Since it is a linear controller, it can be studied in the time domain as well as in the frequency domain. In the time domain, the control signal is equal to

$$s(t) = K_p \left(e(t) + \frac{1}{T_i} \int e(t)dt + T_d \frac{d\,e(t)}{dt} \right) \tag{33}$$

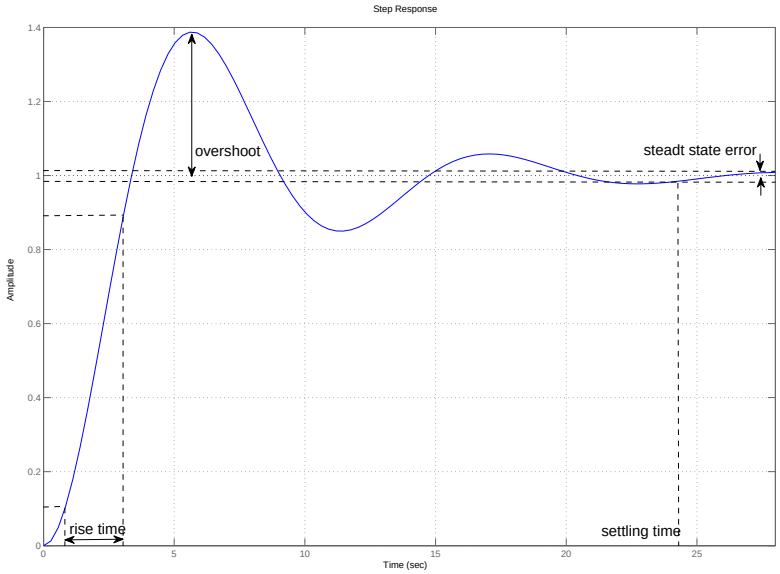

Figure 6: Parameters of a step response

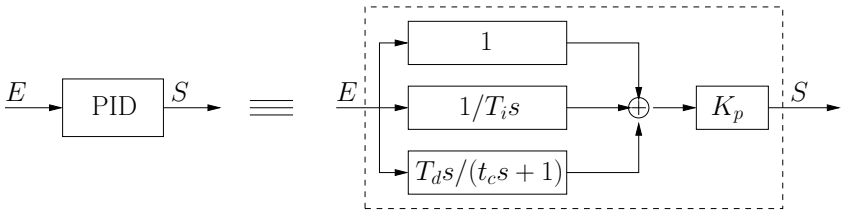

Figure 7: The PID control law

After Laplace transformation, this becomes:

$$S(s) = C(s)E(s) = K_p(1 + \frac{1}{T_i s} + T_d s)E(s) \tag{34}$$

Basically, the controller consists of three terms, each with a typical function. The first term is proportional to the error signal, the second term is proportional to its integral and the third term is proportional to its derivative. However, to be practical, this algorithm needs to be modified, especially for the implementation of the derivative action. Pure derivation is a practical nonsense, as it has unbounded amplification at high frequencies. This is physically impossible and even unwanted. Indeed, the slightest high frequency noise, superimposed on the error signal, would be enormously amplified. Furthermore, the controller output in response to a step function would change annoyingly abrupt (in theory, an impulse arises at the output of the PID).

'Smoothing' can be introduced by processing the error signal with a low-pass filter before taking

the derivative. In practice, both operations are implemented in one filter. The transfer function of the PID-controller is then given as:

$$C(s) = K_p(1 + \frac{1}{T_i s} + \frac{T_d s}{t_c s + 1}) \tag{35}$$

where t_c must be small compared to T_d, to still have a derivative action. When t_c is too large, the third term effectively becomes a proportional term, even for low frequencies. The frequency domain characteristic of the PID controller with and without smoothing is depicted in figure 8.

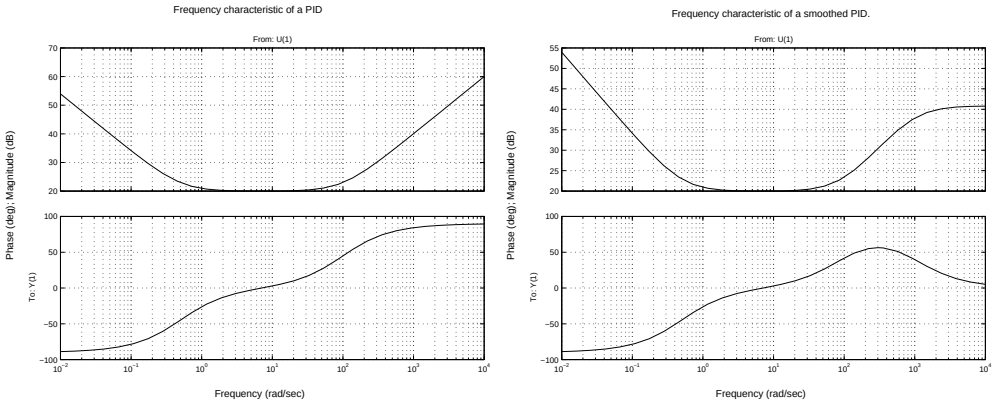

Figure 8: The PID characteristic with and without smoothing for $K_p = 10$, $T_i = 2$, $T_d = 0.01$ and $t_c = 0.001$.

The open loop time response of a PID controller to a step error input, is shown in figure 9. The *P-action* is fast since it reacts immediately to a change in process value. A constant error results in a constant control action due to the P-term. The *I-action* causes the control action to keep on growing as long as the error exists. Therefore the error will always decrease using an I-controller. Compared to the P-controller, the I-controller is rather slow. The *D-action* reacts very fast when an error appears. It is only active at the moment when the error changes.

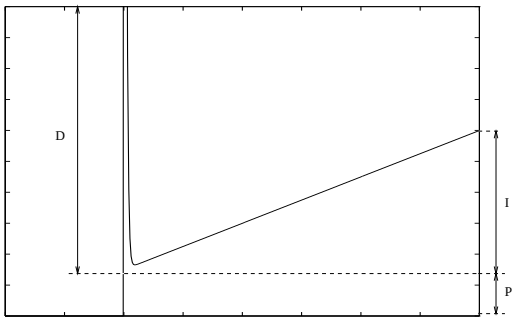

Figure 9: The PID-controller response to a step input in open loop. The P-action results in a constant when the error is constant. The I-action keeps on growing, while D-action is only active at the moment when the step is applied.

Remark: sometimes the parameters of the PID controller are defined slightly different. The end result is however the same: there is a P-action, I-action and D-action. For example, the definition used in the Simulink PID-block is $C(s) = K_p + \frac{T_i}{s} + T_d s$.

Closed Loop performance of the PID-controller

Obviously, PID-controllers will always operate in a closed loop, this is: connected to the system. Basically 3 parameters (excluding t_c) have to be chosen:

- *The gain* K_p
- *The integration time or reset time* T_i
- *The derivation time or lead time* T_d

We will now analyze in more detail the effect of the different options.

P-action
If we assume a system with a DC-gain equal to K and a P-controller with gain K_p, one can readily calculate that in a closed loop configuration (see figure 10) after the controller transients, the process output will not converge to the reference r: a steady state error remains (steady state error = the error of the output in comparison with the desired input when the transient response has decayed). An equilibrium will be found between output y and error e.

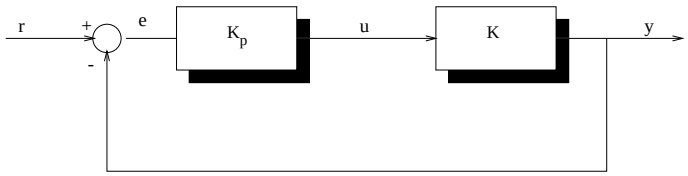

Figure 10: Static equilibrium between error and output with a proportional controller.

At equilibrium one obtains:

$$y = K_p K e, \quad e = \frac{1}{1+K_p K} r, \quad y = \frac{K_p K}{1 + K_p K} r \tag{36}$$

Notice that this steady state error decreases with increasing DC-gain K or K_p. Only with an infinite DC-gain K or K_p, the error will be zero. This is also achieved by adding an integrator to the controller. An integrator contains a $1/s$-factor such that no offset will occur since the DC-gain of the system is ∞. To conclude, a DC-offset between the reference value and process output will always exist with a pure P-controller, but can be made small by choosing K_p large.

However, when a step input is applied to a closed loop system with a very large K_p, the controller will respond very fast to any error with an almost full control action. Hence, taking K_p too large may result in overshoot and oscillations. The system can even become unstable (see section 2.2).

A trade-off has to be made between overshoot and DC-offset. In figure 11 this effect is shown for a plant with transfer function $P(s)$ with a proportional controller. A step with unitary magnitude is applied to the closed loop system with unity feedback.

$$P(s) = \frac{1}{s^2 + s + 1}$$

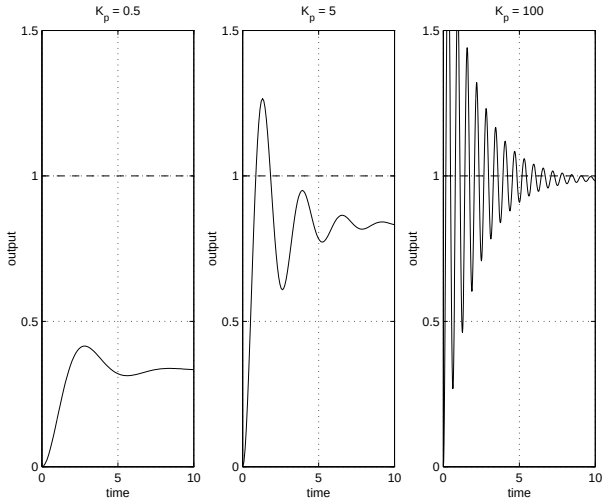

Figure 11: Effect of gain K_p on a response to a unit step in reference value. When $K_p = 0.5$, the steady state error is large. With $K_p = 5$, the steady state error is reduced, and there is some oscillation. With $K_p = 100$, nearly no steady state error, but too many oscillations.

I-action

As already mentioned, the P-action alone results in a steady state error. This can be avoided by introducing an infinite DC-gain, realized by an integrator. The integrator action is thus added in order to *avoid DC offset.*

Since the I-action is not changed whenever the error is zero (an integral remains constant when the input is zero), there is a risk for overshoot. When the output value crosses the reference value, the I-action is maintained (integral!). Thus the I-action will continue to act in the same direction, which results in overshoot. The I-action will decrease subsequently due to the negative error, until it is zero. The resulting oscillation may have a relatively large period. See also figure 12. Since a short integration time T_i results in a strong I-action, a small T_i results in a more oscillatory behaviour.

PI-action

Compared with the P-controller, the PI-controller eliminates the steady-state error. Compared with the I-controller, the PI-controller is faster with a smaller oscillation period.

PD-action

Contrary to the I-action, the D-action is most active when the system output crosses the reference value (for unity feedback). Since it is proportional to the derivative of the error, it can 'predict' how the error will change in the near future, and will act accordingly. Therefore it has a *damping*

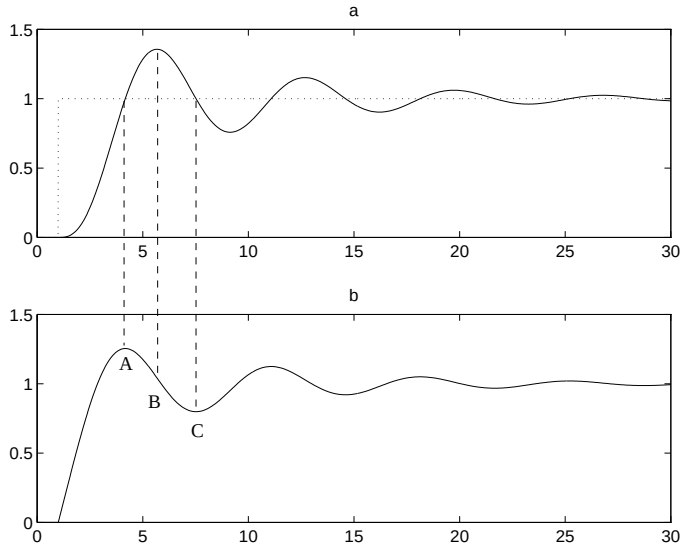

Figure 12: Effect of an integrator control action. a. The reference value and the process output. b. The control action. In point A the error becomes negative, nevertheless the I-action is at its maximum value! In point B the error is maximally negative, the integral is changing quickly. In C the error is zero again and will become positive, thus the integral will grow again... Note the 90^o phase difference between the error signal and the I action.

effect and one can choose a larger K_p value, which will result in less offset. The larger T_d, the stronger the D-action compared to the P action.

PID-action
The I-action is added to eliminate steady-state error, the P-action to make the response faster, and the D-action is added to provide damping.

2.2 Stability analysis

Bode plots are good tools to analyze the robustness of a closed loop system, i.e. to determine how stable a system is. Robustness can be expressed as *gain margin* and *phase margin*.

- The *gain margin* (GM) gives the allowable gain variation of the plant and the controller before the closed loop system becomes unstable. It is the increase in the loop gain ($= C(s)P(s)$ for unity feedback) that will result in a marginally stable system, and is hence calculated at frequencies where the phase is -180^o (figure 13). As a rule of thumb, the GM should be at least 3 dB.
- The *phase margin* (PM) gives the allowable phase variation (or time delay) before the closed loop system gets unstable, and is hence calculated at frequencies where the gain is 0 dB (figure 13). As a rule of thumb, the PM should be at least 60^o.

REMARK: the phase and gain margin give information about the CLOSED LOOP system, but are calculated on the LOOP GAIN!

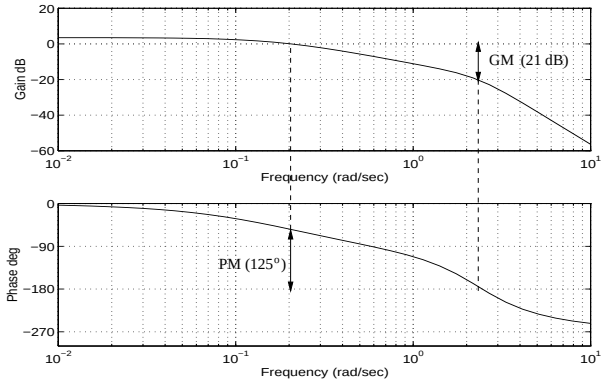

Figure 13: Phase margin and gain margin.

The impact of the different PID-actions is shown in the next figures.

P-control

In the case of proportional control, the controller transfer function is a gain:

$$C(s) = K_p$$

and the control signal is proportional to the control error. From the Bode plot of a proportional controller (figure 14), it follows that the loop gain is multiplied by K_p, while the phase shift is not influenced at all.

The purpose of proportional control is to increase the loop gain, in order to obtain a higher accuracy (DC-gain). However, due to stability requirements (GM decreases when K_p increases), K_p is limited. For plants with a tendency to oscillate, one can easily verify that the rise time T_r decreases, but that the settling time T_s and the overshoot often increase.

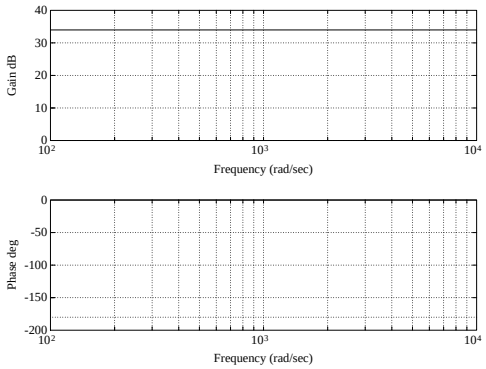

Figure 14: The Bode plot of a P-controller ($K_p = 50$).

I-control

In the case of integral control, the controller transfer function is an integrator:

$$C(s) = \frac{K_p}{T_i s}$$

and the control signal is proportional to the integral of the control error. From the Bode plot of an integral controller (figure 15), it follows that the loop gain is multiplied by $\frac{K_p}{T_i}$ at $\omega = 1$ and that an additional phase shift of -90^o is introduced.

> The purpose of integral control is to make the loop gain at DC equal to ∞, in order to obtain a zero steady state error. However, the additional phase shift (PM decreases) forces the designer to decrease the bandwidth, to ensure relative stability (so $\frac{K_p}{T_i}$ is limited).

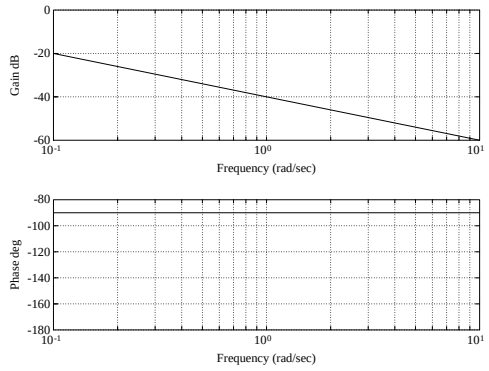

Figure 15: The Bode plot of a I-controller ($K_p = 1, T_i = 100$).

D-control

In the case of smoothed derivative control, the controller transfer function is a low-pass filtered derivative term:

$$C(s) = \frac{K_p T_d s}{t_c s + 1}.$$

From the Bode plot of a derivative controller (figure 16), it follows that the loop gain is multiplied by $K_p T_d$ at $\omega = 1$ and that an additional phase shift of $+90^o$ is introduced.

> The purpose of derivative control is to increase the response speed (decrease T_r). However, due to the amplification at high frequencies (noise sensitivity), the derivative action must be limited.

PI-control

In PI-control, the advantages of P- and I-control are combined. P-action is added for bandwidth and speed, the I-action is added to guarantee a zero steady state error. From the Bode plot of a PI-controller (figure 17), it follows that a high loop gain is introduced for lower frequencies, while the additional phase shift vanishes for higher frequencies. The positive and the negative effects of

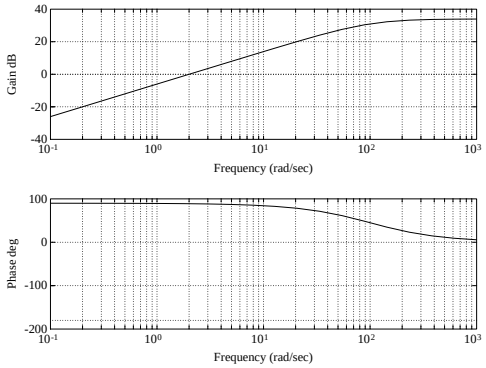

Figure 16: The Bode plot of a D-controller ($K_p = 1,\ T_d = 0.5,\ t_c = 0.01$). The derivative action is bounded for frequencies $\omega \geq \frac{1}{t_c} = 100$.

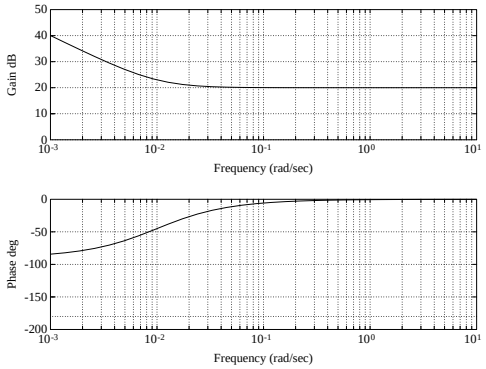

Figure 17: The Bode plot of a PI-controller ($K_p = 10,\ T_i = 100$). The integral action is higher for frequencies $\omega \leq \frac{1}{T_i} = 0.01$.

PI-control become more pronounced by making the integration time T_i smaller: the high gain for low frequencies increases, but the phase margin decreases.

PD-control

In PD-control, the advantages of P- and D-control are combined. P-action is added for bandwidth and speed, the D-action to further decrease the rise time (while P is limited by stability requirements). From the Bode plot of a PD-controller (figure 18), it follows that a high phase lead and a high gain are introduced for higher frequencies: the speed of response increases, together with the sensitivity to noise. The positive and the negative effects of PD-control become more pronounced by making the differentiation time T_d larger: the phase lead for high frequencies increases, but the noise sensitivity also increases!

PID-control

An attempt to combine all previous advantages, while accepting the corresponding disadvantages, is the PID-controller. It is easy to understand that the relative sizes of the different controller parameters determine which effects become more/less important. By analyzing the Bode plot of the PID-controller (figure 19), it is possible to do some predictions on the closed loop behaviour.

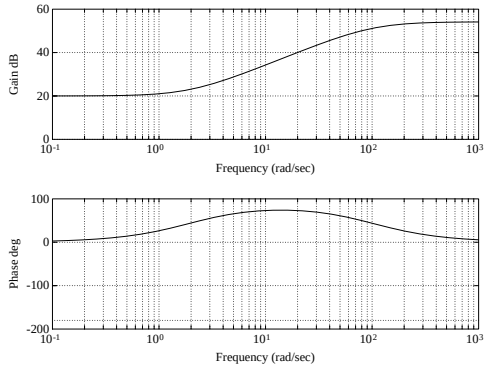

Figure 18: The Bode plot of a PD-controller ($K_p = 10,\ T_d = 0.5,\ t_c = 0.01$). The derivative action is bounded for frequencies $\omega \leq \frac{1}{T_d} = 2$ or $\omega \geq \frac{1}{t_c} = 100$.

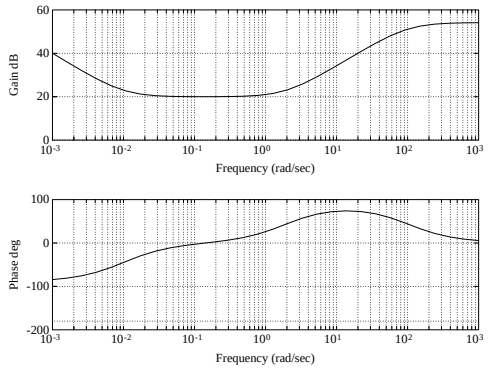

Figure 19: The Bode plot of a PID-controller ($K_p = 10,\ T_i = 100,\ T_d = 0.5,\ t_c = 0.01$).

2.3 Parameter tuning

Now that we know the effects of the PID parameters, we have all the ingredients to design a PID controller for a feedback system such that the closed loop response meets the specifications. These specifications are usually expressed in terms of a maximum steady state error that is allowed, a maximum overshoot and a maximum settling time.

When tuning your PID parameters, it is helpful to bring some structure into your search quest:

- *Start with a pure P-action*
  From the Bode plot of the open loop system, you can see how much gain you can provide before the stability margins become too small. Try to give as much gain as possible since this will make your controller fast. If you find a gain for which your step response is fast enough and the steady state error small enough, you can stop here.

- *Add a I-action to your controller to decrease steady state error*
  By adding integration to your controller, you can make the steady state error go to 0. Since this adds extra gain to your controller at low frequencies, it will also make your controller

faster. Start with a large T_i (weak I-action) and increase until your system becomes unstable. Because of your already strong P-action, this may happen fairly quickly. In that case you can lower your P-action in order to allow a stronger I-action. Keep an eye on your open loop Bode plot: I-actions will decrease your phase margin!

- *Add a D-action to quickly react to disturbances and/or dampen the response*
  If your PI-controller has to allow a lot of overshoot in order to achieve a fast enough response, you can add a D-action to dampen the response. Due to the phase lead of the D-action, the phase margin can be increased. Start with a small D-action and increase until you get the desired response. Since the D-action adds gain to high frequencies, your response will also get faster, but you run into the risk of amplifying noise that is present in the system. Therefore, if you can avoid a D-action it is better not to include it. If you do include the D-action, take precautions by limiting the amplification on very high frequencies by including t_c after you found a suitable T_d.

Tuning the parameters of a PID controller is an iterative process. It is highly recommended to take notes during this process!

3 Control system for the Dragon

In section 1, a model was derived for the behaviour of the Dragon. Our goal is now to design a control structure that keeps the Dragon upright, even in the presence of disturbances. A second objective is to allow the Dragon to drive around without falling over.

To design the control system, we will simulate it in Matlab and/or Simulink. In Matlab you are limited to the analysis of linear systems, but the advantage here is that you can easily generate Bode plots, pole-zero plots,... Simulink can deal with both linear and nonlinear systems, but there is no functionality to make Bode plots etc.

Have a look at the following Matlab functions using '`help`'. They will come in handy when analyzing linear (feedback) systems:

Function Name	**Function Description**
tf	Creation of transfer functions
series	Computes a series system connection (you can also multiply transfer funtions)
feedback	Computes the feedback interconnection of two systems
pole	Computes the poles of a system
tzero	Returns the transmission zeros of a system
pzmap	Plots the pole-zero map of a linear system
bode	Generates Bode frequency response plots
dcgain	Calculates the DC gain of a system
margin	Computes gain margin, phase margin and crossover frequency
step	Calculates the unit step response of a system
lsim	Computes time response of a system to an arbitrary input and initial conditions
ltiview	Interactive user-interface for analysis of transfer functions

When implementing your control system in Matlab/Simulink, use symbols instead of values as much as possible and make a file where you assign the values to the symbols. By running that file, both Matlab and Simulink will know the value corresponding to the symbol. If you work like this, you can easily make changes and test the effect of changing parameters.

3.1 Open loop analysis

Before you start implementing a control structure, it is a good idea to analyze the process/device in open loop in order to get a feel of how the system behaves. It allows you to see what the contribution of the controller is to the behaviour of the system afterwards.

Implement the model of the Dragon in Matlab using transfer functions. Use `ltiview` *to check the behaviour of the open loop system. You can do a time domain analysis by looking at the step response and a frequency domain analysis by looking at the Bode plot and the pole-zero plot.*

The Dragon has one input (the voltage applied to the motor) and two outputs (the speed of the motor and the inclination of the Dragon).

- Do a time domain analysis with the speed of the motor as the output. What is the interpretation of this step response? Determine the rise time and the settling time. What is the percentage overshoot? Can you say something about the steady state error?
- Do a frequency domain analysis for this system. Look at the Bode plot and the pole-zero plot.
- Do a time domain analysis with the inclination of the Dragon as output. What is the interpretation of this step response? Is this system stable?
- Do a frequency domain analysis of the inclination by looking at the Bode plot and the pole-zero plot. Do these views allow you to determine the stability of the system?
- Make a Simulink scheme of the Dragon. Apply a step in Simulink and check the response with the one from Matlab.

3.2 Speed/Current control system

When the Dragon is balancing itself, the trick is to make sure its axle is directly under the center of gravity. When the Dragon starts tipping over, the balance control system thus will have to apply a voltage to the motors such that they start driving towards the direction in which the Dragon is tipping over. Due to the acceleration, a torque will be applied to the base of the Dragon allowing it to couteract the tipping torque.

It is important to note that there are 2 motors, each connected to a wheel. The balancing control system will steer both wheels in the same way. We have to make sure that both motors react identically to the steering signal and have the same speed and acceleration.

- What will happen when there is a small rock in front of one of the wheels? Will this wheel have the same speed as the wheel without a rock in front of it? (Or when one of the wheels is on a surface with more friction than the other wheel?)

- What will happen when there is a small difference between both motors due to different wear of the gearbox? Or differences due to the production process?

To cope with these disturbances and process deviations, a feedback control system can be applied. Design a PI control system that makes sure that the speed/acceleration of a wheel can be accurately controlled, independent of disturbances and process deviations. There are two options to do this: control the speed of the motors or control the current through the motors (which is propotional to the acceleration). It should be noted here that the speed of the wheels is measured with encoders that have a rather course quantisation and some delay, making it hard to get an accurate speed control system. Speed controlled wheels later on also result in a steady state error in the balancing control system that cannot be eliminated. The current through the motors can be accurately controlled and this results in a steady state error for the balancing control system that can be made zero (try to explain this once you are designing the balancing controller). It is however up to you to choose wich control structure you want to use.

- Draw a block diagram of your control system.
- Implement the transfer function of this control system in Matlab and make a Simulink scheme of it. In Simulink you can find a PID block under '`Simulink Extras -> Additional Linear`'. Both a standard and a smoothed version are available.
- Set the parameters of the PI controller such that you get a fast and accurate control system, preferably with little or no overshoot. Since in the end the balance controller will control the desired speed/acceleration of the weels, the speed/current controller should be at least an order of magnitude faster than the balance controller.

Hint: it may be helpful to set the gravity $g = 0$ during initial simulations. This way you don't get side effects from your Dragon falling over. Once you get close to reasonable PI parameters, set $g = 9.81$ again to finetune the parameters.

Hint: you want to make this controller as fast as possible. However, the control system of the Dragon will be implemented digitally (discrete time) while your design is done in continuous time. Depending on your maximum sampling frequency, you are limited in the bandwidth for your control systems and thus their speed. Make sure you can actually implement the control systems you design! (If your fastest control loop behaves the same in discrete time as it does in continuous time, you should be fine. So it might help you sleep when you can at least verify this by simulation.)

The following points can be helpful when setting up a discrete simulation of your Dragon:

- Simulink can do mixed-mode simulations (continuous time and discrete time in the same model). There is no need to discretize your Dragon model. This also reflects what is happening in real life: only your control system is discrete, the Dragon itself is still continous time! At the interface between the continuous part and the discrete part you will need to place a '`Discrete -> Zero-Order Hold`' block to go from continuous to discrete. This will allow you to set the sampling rate. To go from discrete to continous, no extra block is required.
- You will have to build your own PID controller in discrete time. Use a $1/z$ block as a 'memory cell' to implement a first order integrator (output is previous output plus current input) and differentiator (output is difference between input and previous input).

3.3 Balancing control system

Now that we can control the wheels accurately and identically, we can start the design of a control system to keep the Dragon upright. When the Dragon starts tipping over to the right, the control system should steer the wheels such that we get an acceleration to the right. Due to the acceleration a torque will be applied to the base of the Dragon, allowing it to counteract the inclination.

Keeping the Dragon under some inclination $\theta \neq 0$ requires a constant acceleration of the Dragon. This cannot be sustained very long due to the fact that the supply voltage is limited and thus the current through the motor is limited. Therefore, the steady state error has to be very small. Secondly, the Dragon has to react very quickly when it starts falling over. Your controller will need a lot of gain and possibly a D-action to react very quickly to a disturbance. For your simulations, you can use a non-smoothed PID controller ($t_c = 0$) since noise is not modeled. Once you get closer to the actual implementation, keep in mind that you may need to add smoothing.

- Extend your model of the Dragon such that it has 2 wheels.
- Draw a block diagram of a control system to control the inclination of the Dragon and implement it in Simulink.
- Set the parameters of this control system such that the Dragon can keep itself upright.
  - Design the parameters based on the step response of the closed loop system
  - Keep an eye on the open loop Bode plot to ensure that you have sufficient gain and phase margin
  - When you found suitable parameters, test that the Dragon can keep itself upright. Extend your model to include the effect of a disturbance to the inclination (someone shortly pushes the top of the Dragon).
  - Keep an eye on the voltage and current required to keep the Dragon upright.
- Once you have found parameters for the control system, you can extend your Simulink model and include the effect of a limited power supply.
- To include nonlinearities in your model, you can implement equations (20-21) in Simulink. Figure 20 can be used as a starting point. *At this point in the project, it is probably best to use a non-linear model in Simulink (keep using the linear one in Matlab to generate Bode plots!!). In the past, we had some numerical problems with the linear model due to the implementation of the transfer function block by Simulink, resulting in 'weird' effects. This will not happen when you build your own nonlinear model. If you prefer a linear model, implement your own in a similar way as the nonlinear one, using figure 20 and equations (24-25). Do not use the '*`Transfer Fcn`*' block anymore in the mechanical part of your Dragon model!*

When you can make your virtual Dragon balance, check the current through the motors in a realistic scenario. If this current exceeds the maximum current allowed by either the motors or the H-bridge driving them, you will have to come up with a solution to limit this current. A power resistor in series with the motor could be helpful, but it may have an influence on the parameters of your controllers...

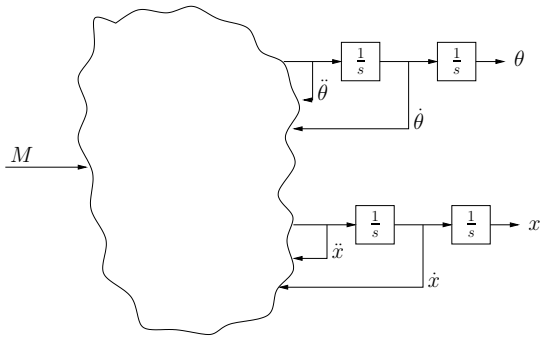

Figure 20: Fill in the cloud to complete the nonlinear model of the Dragon

3.4 Steering the Dragon

Keeping the Dragon upright is only part of the problem. We also want to be able to drive the Dragon around. This means that, on average, the wheels should have some constant speed. Devise a structure to drive the Dragon around. Start with the simple problem of driving forward and backward, without worrying about making turns:

- What would happen when you apply an offset on the desired speed/current of the motors?
- What would happen if you force the Dragon to balance under a certain angle?

To make turns, the speed of the wheels of the Dragon should be different from each other. Depending on how you are going to implement steering commands, you will need a slightly different implementation of your steering control systems.

In any case, the control systems steering the Dragon should be very slow: the priority of the Dragon is to not fall over!

3.5 Being accurate in your simulations

The reason for all these simulations is that in Matlab/Simulink we can to all kinds of analyses, most importantly look at Bode plots. Once you are working on real hardware, you are much more limited in what you can 'see'. In order to be of any value, your simulation must model your hardware as close as possible (within reason). Any amplifier, anti-aliasing filter,... should be modeled. Take our word for it that figuring out that a filter is messing up your phase margin by looking at the hardware is extremely difficult (and frustrating). If you check this on a Bode plot in Matlab, you will immediatly find the problem and even the solution!

Appendix: Alternative derivation of the motion equations

To derive the motion equations of a rigid body like the Dragon, Lagrangian mechanics can be used. The motion equations are then

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = \text{external force} \tag{37}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = \text{external moment} \tag{38}$$

with the Lagrangian L defined as

$$L = \text{kinetic energy} - \text{potential energy} \tag{39}$$

The Dragon can be viewed as 2 independently interconnected parts: the weels and the motors+body+battery. To determine the kinetic and potential energy of the Dragon, we need to know the position and speed of both parts. The horizontal speed $\dot{x}$ of the wheels will be measured by the control system, along with the angle θ of the Dragon. The remaining positions and speeds can be calculated as

$$x_z = -l_z \sin(\theta) + x \tag{40}$$

$$y_z = l_z \cos(\theta) + r \tag{41}$$

$$\dot{x}_z = -l_z \cos(\theta)\dot{\theta} + \dot{x} \tag{42}$$

$$\dot{y}_z = -l_z \sin(\theta)\dot{\theta} \tag{43}$$

with $\{x_z, y_z\}$ the horizontal and vertical position of center of gravity of the motors+body+battery. The potential energy $P = mgh$ of the Dragon is then

$$P = m_w g r + m_z g \left(l_z \cos(\theta) + r\right) \tag{44}$$

The kinetic energy $K = \frac{mv^2}{2} = \frac{I\omega^2}{2}$ of each part of the Dragon can be decomposed into a part due to linear motion of the center of gravity and a part due to rotation arout the center of gravity:

$$K = \left[\frac{m_z\left(\dot{x}_z^2 + \dot{y}_z^2\right)}{2} + \frac{I_z\dot{\theta}^2}{2}\right] + \left[2 \times \frac{m_w\dot{x}^2}{2} + 2 \times \frac{I_w(\dot{x}/r)^2}{2}\right] \tag{45}$$

Applying equations (44) and (45) to (39) and noting that the external force and moment induced by the (two) motors when driving forward are respectively $2\frac{M}{r}$ and $2M$, the motion equations (37-38) become

$$-m_z l_z \cos(\theta)\ddot{\theta} + m_z l_z \sin(\theta)\dot{\theta}^2 + (m_z + 2I_w/r^2 + 2m_w)\ddot{x} = \frac{2M}{r} \tag{46}$$

$$(I_z + m_z l_z^2)\ddot{\theta} - m_z l_z \cos(\theta)\ddot{x} - m_z l_z g \sin(\theta) = 2M \tag{47}$$