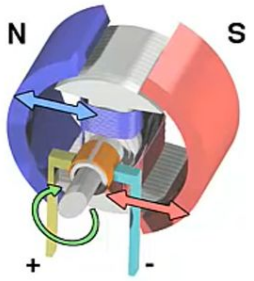
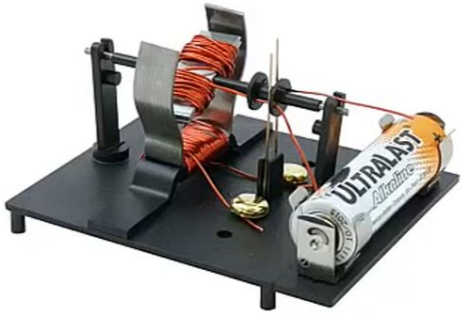
Motors and controllers

DC Motors

- Stationary permanent magnet
- Electromagnet on axis induces torque
- Split ring + brushes switch direction of current
- Maybe you built one in school



CC BY SA by Wapcaplet
http://en.wikipedia.org/wiki/File:Electric_motor_cycle_2.png



http://www.hometrainingtools.com/dc-motor-kit/p/EL-KIT02/

http://www.seeedstudio.com/depot/bitcraze-m-64.html?ref=side

When the electromagnetic is switched on, it turns using a torque on axis and this makes the axis spins to align the magnetic field of the electromagnets with the permanent magnet on the outside.

Control of DC Motors

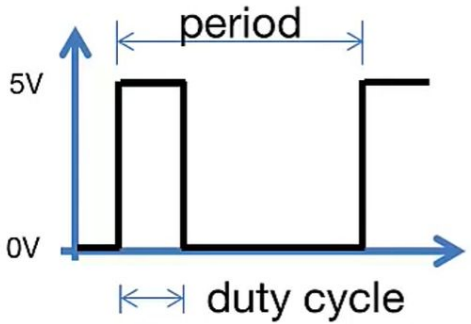
To control the speed of the DC motor, you have to adjust the power. If you give voltage to it, it will start spinning faster.

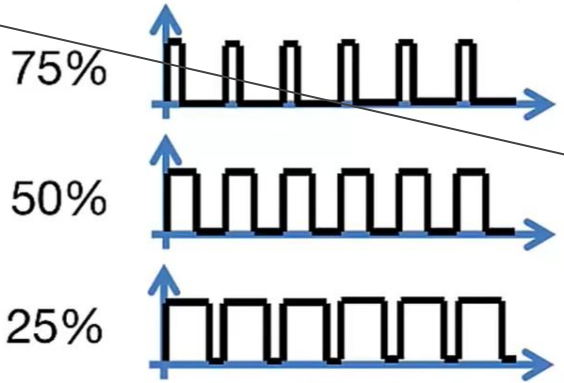
- More power = faster rotation
- How to modulate power using a digital signal?
- Pulse width modulation (PWM)
- Duty cycle = proportion of on time vs. period

As it takes the speed is controlled by a computer or microcontroller which means that it takes a digital signal

A common approach implemented in MCUs

The idea is that you have a digital signal that you switch on and off very simply at a certain frequency.





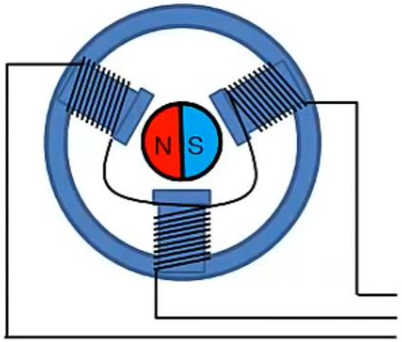
Duty time ⟶ the amount of time that the signal is switched on during this period. The larger duty cycle, the more power transmits.

Note that: These signals need to be amplified using a *MOSFET* to generate the high current we need for powering the motors.

Brushless Motors (An alternative for DC motors)

- Electromagnets are stationary
- Permanent magnets on the axis (either inside or outside)
- Three coils (or more)
- No brushes (less maintenance, higher efficiency)

Between the coins (inside) or around the coins (outside)



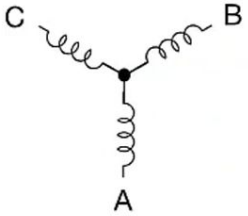
https://www.hobbyking.com/hobbyking/store/__25556__AX_2810Q_750KV_Brushless_Quadcopter_Motor.html

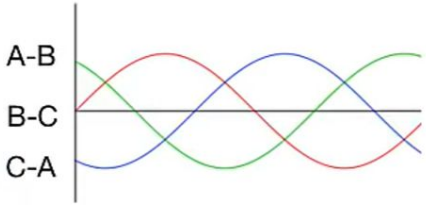
No brushes → No need to replace them → **less maintenance**.

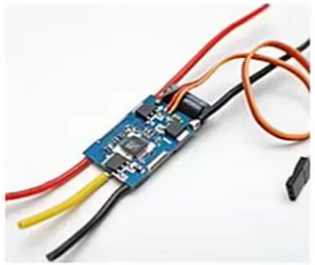
Higher efficiency → You can regulate the power better than you give to electromagnets.

Brushless Controllers

- Typically one microcontroller per motor
- Generates PWM signal for the three motor phases
- AC signal converter (capacitor+MOSFET) to convert PWM to analogue output
- Measure motor position/speed using back-EMF







http://en.wikipedia.org/wiki/File:ESC_35A.jpg

How does brushless motor work?

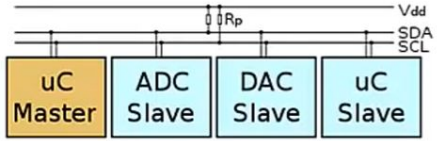
You set a certain current between two coils. For example between A and B or B and C.

The signal between the pairs of coils is shifted by 120°. That induces a rotary magnetic field on the coils and this makes the axis of the brushless motor spins at the frequency of these three signals.

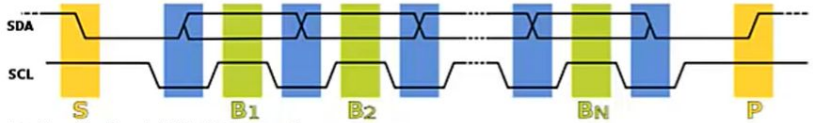
The good thing in brushless motors is that you can measure the position and the speed of the motor by evaluating the electromagnetic response.

I²C Protocol

- Serial data line (SDA) + serial clock line (SCL)
- All devices connected in parallel
- 7-10 bit address, 100-3400 kbit/s speed
- Communication between motor controller and autopilot

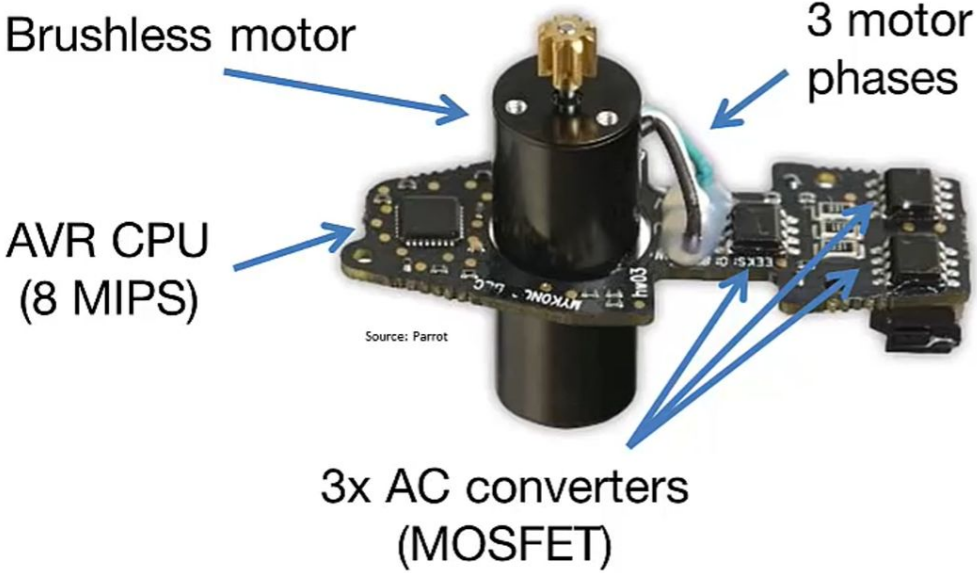


http://en.wikipedia.org/wiki/File:I2C.svg



http://en.wikipedia.org/wiki/File:I2C_data_transfer.svg

Example: Parrot Ardrone

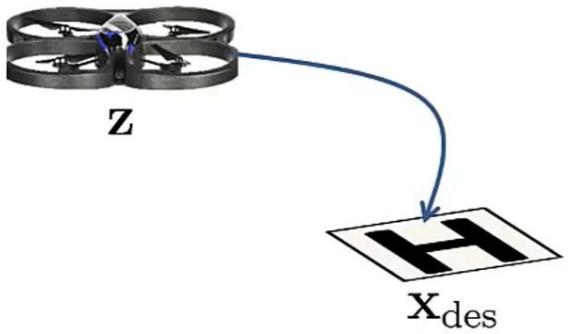


http://droneflyers.com/category/ar_drone/

Feedback control

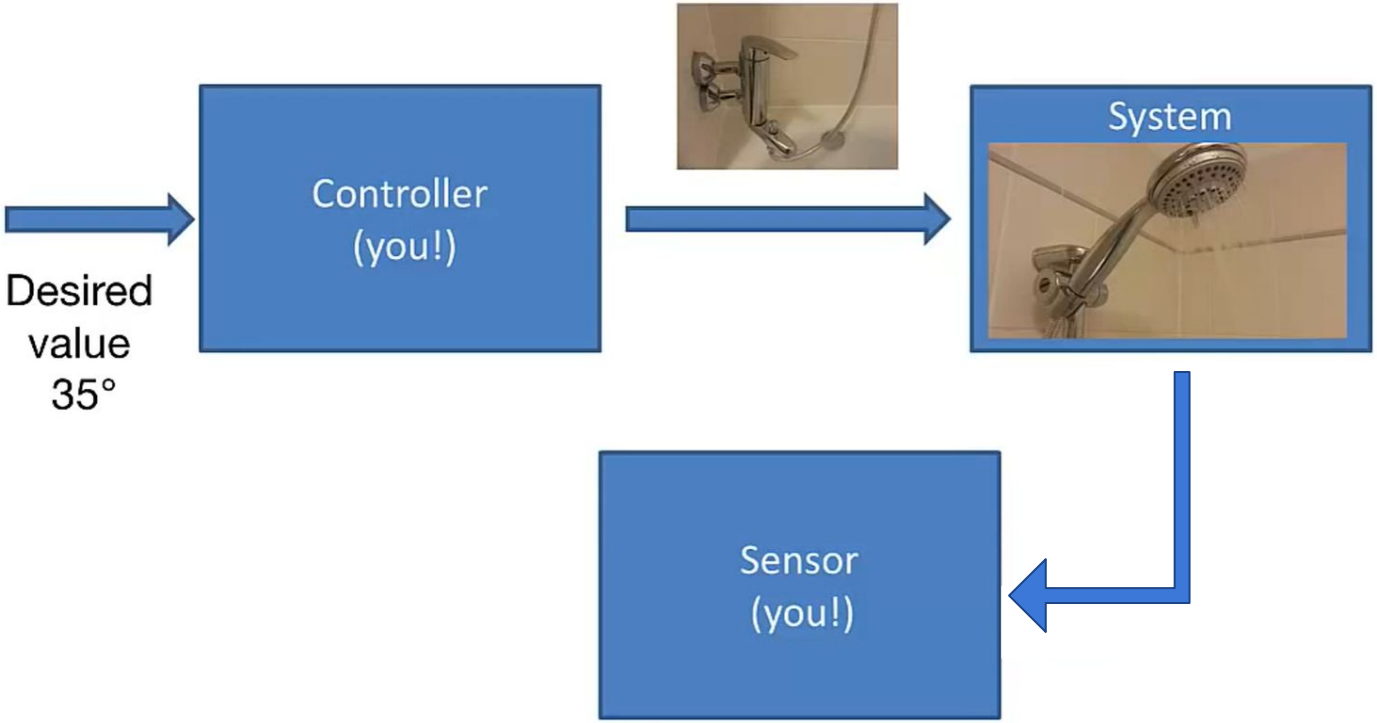
Motivation: Position Control

- Move the quadrotor to a desired location $\mathbf{x}_{\text{des}}$
- How can we generate a suitable control signal $\mathbf{u}$?
- Current location (observed through sensors) $\mathbf{z}$

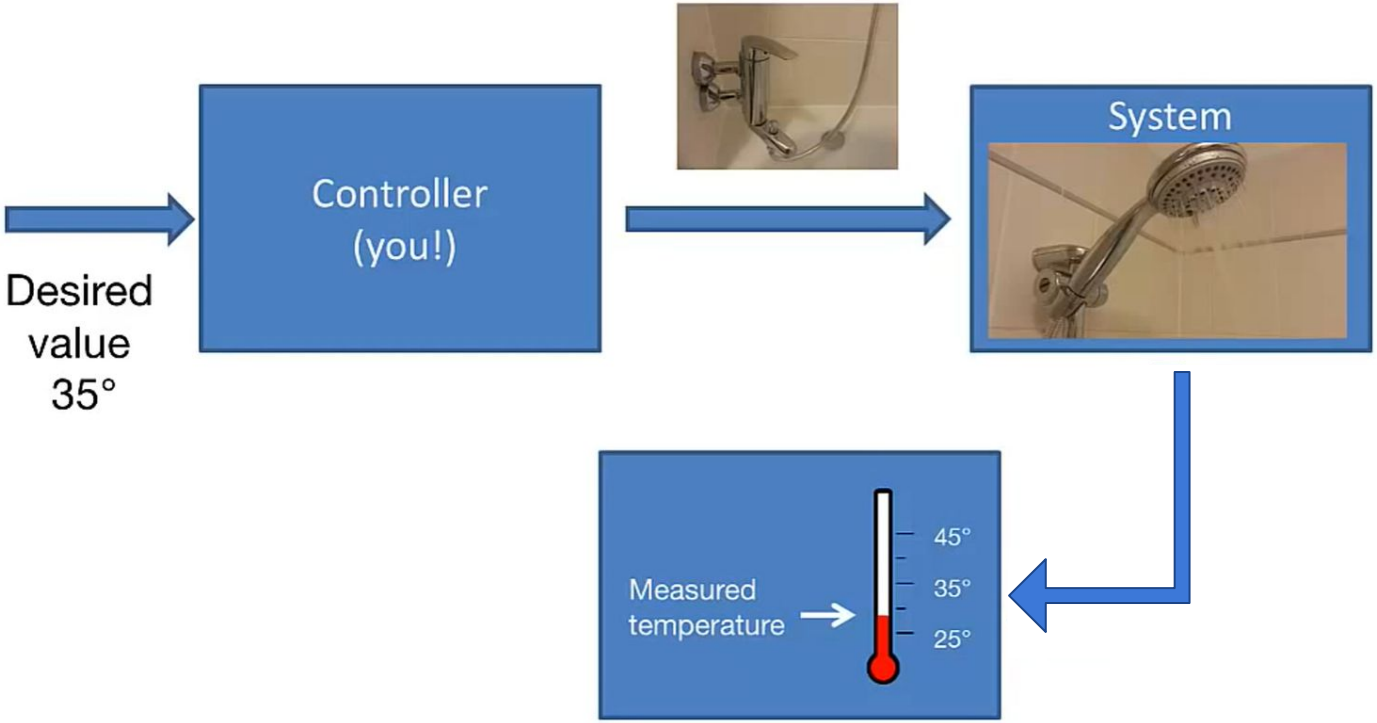


We can know the current position of the quadrotor that we observe from the sensors somehow and capture it with the goal position.

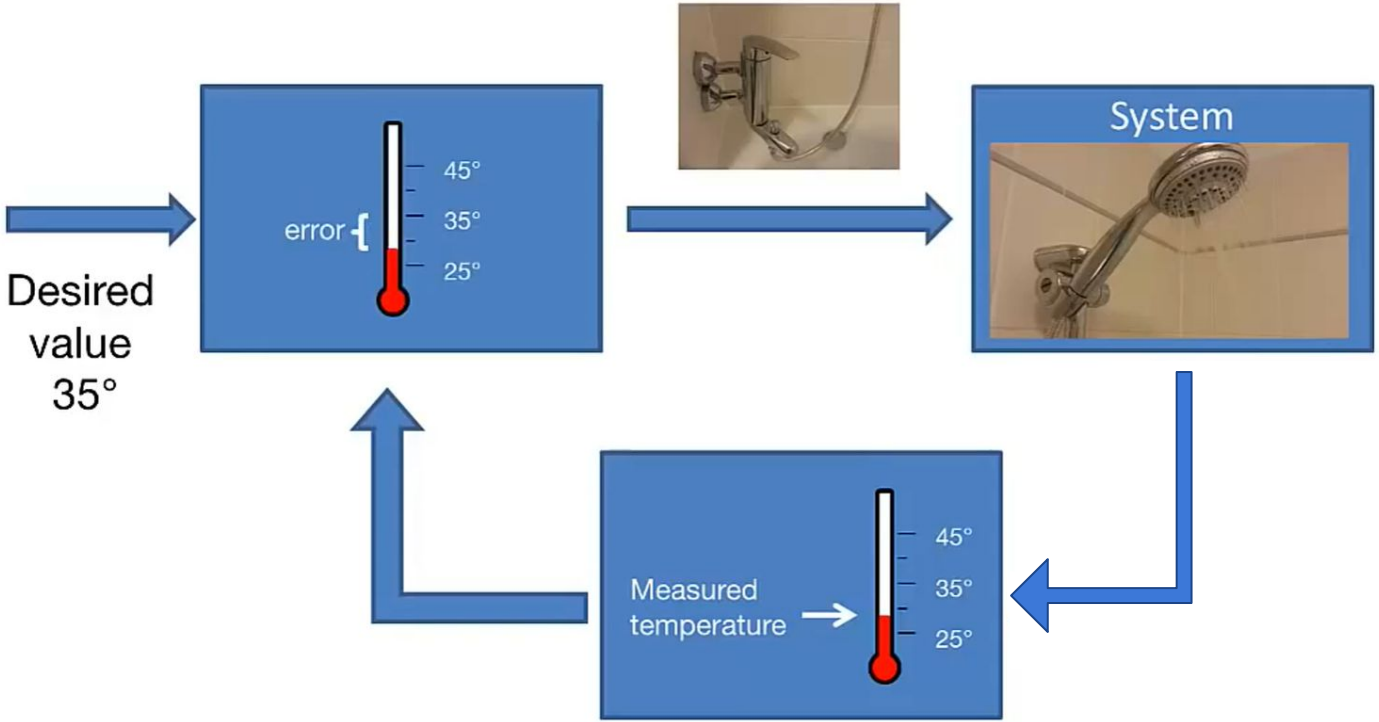
Feedback Control – Generic Idea



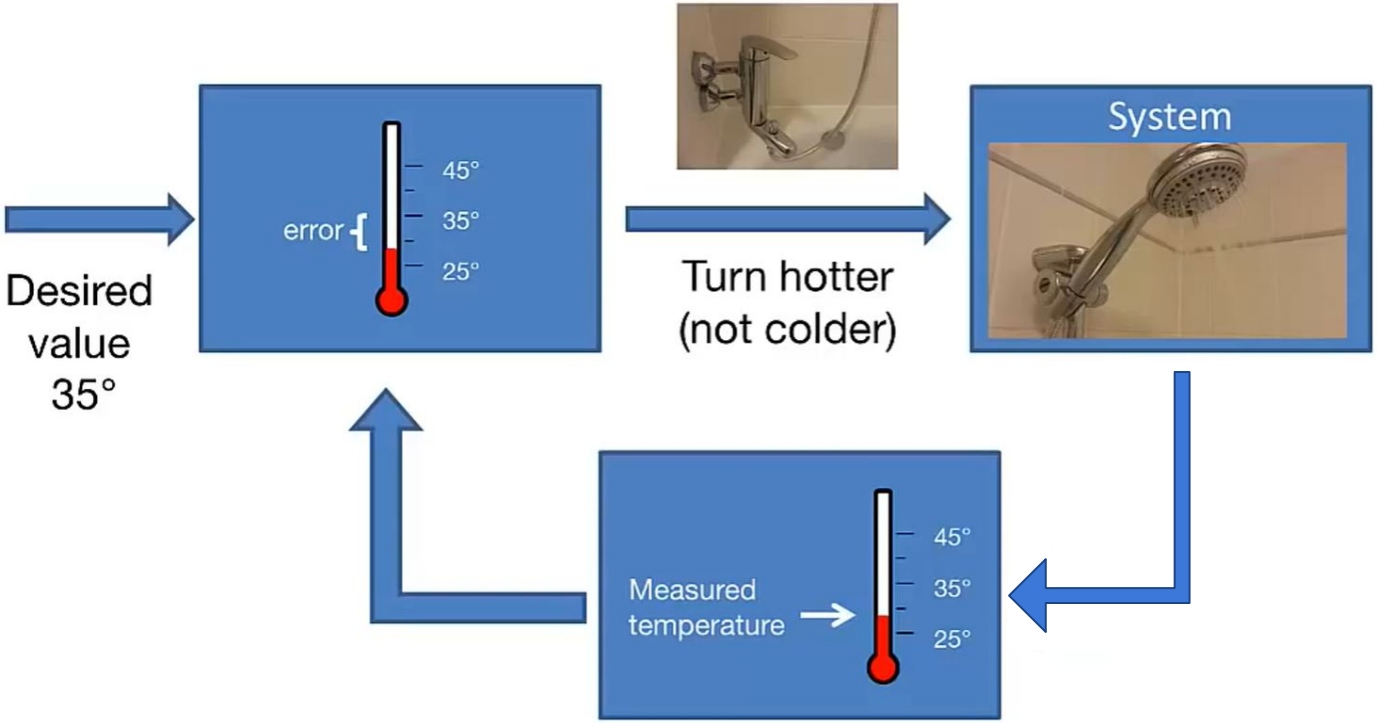
Feedback Control – Generic Idea



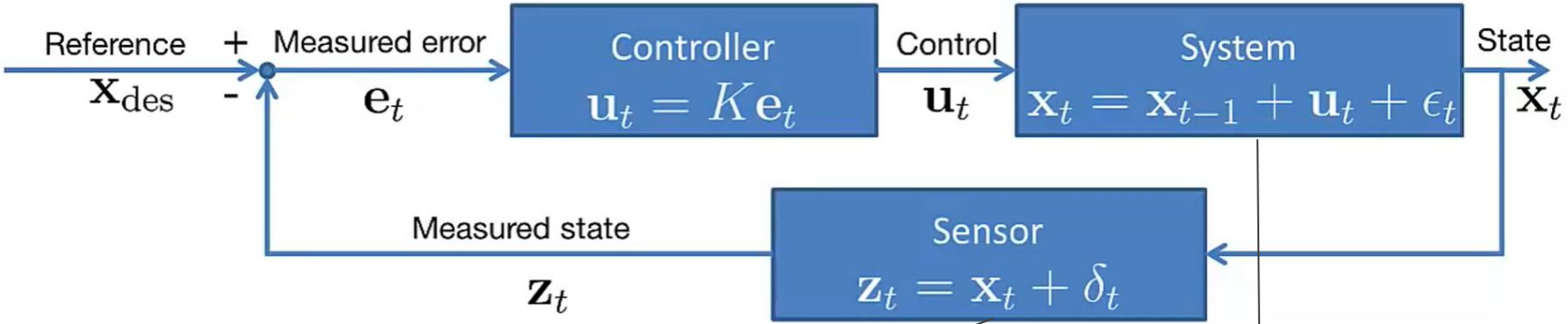
Feedback Control – Generic Idea



Feedback Control – Generic Idea



Feedback Control – Block Diagram



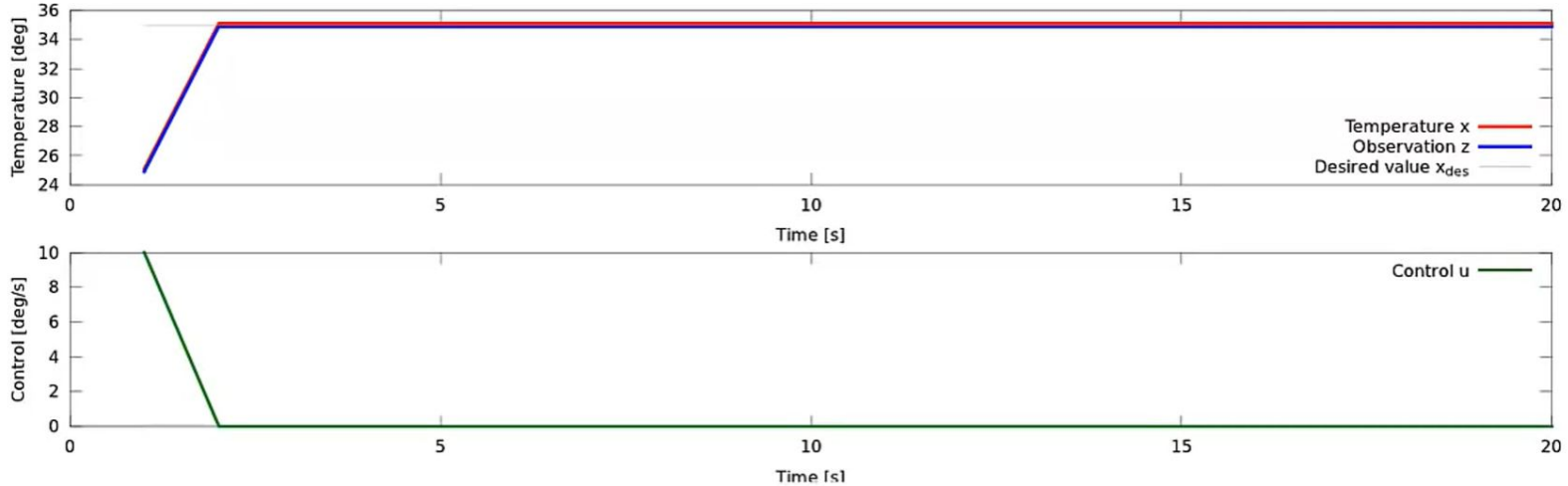
X_t [new state] = X_t-1 [Old state] + U_t [Controller output] + ε_t [epsilon, error constant, system noise]

Sensor gives us the observed state
Z_t = X_t (system temperature) + δ_t (delta, noise, distributed according to some distribution)

Proportional Control

- P-Control: $\mathbf{u}_t = K\mathbf{e}_t$

U_t: Controller output
K: Gain(+ve) constant
e_t : Measured error



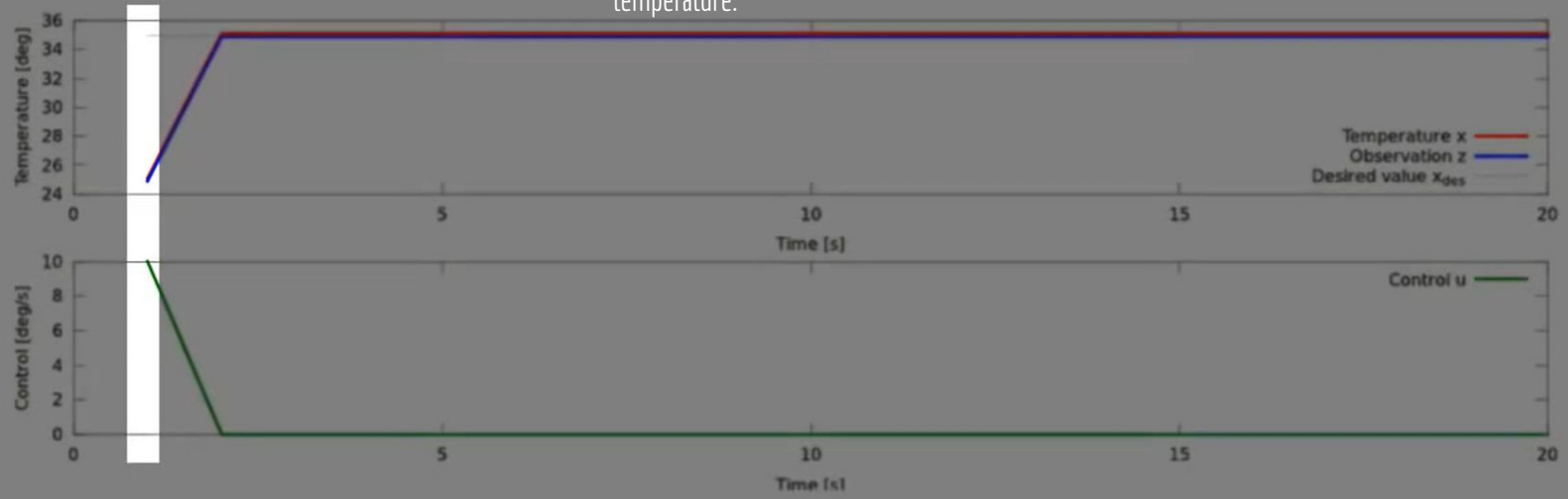
Proportional Control

- P-Control: $\mathbf{u}_t = K\mathbf{e}_t$

If k = 1:

Assume we start at 25° temperature and we want to achieve 35° temperature.

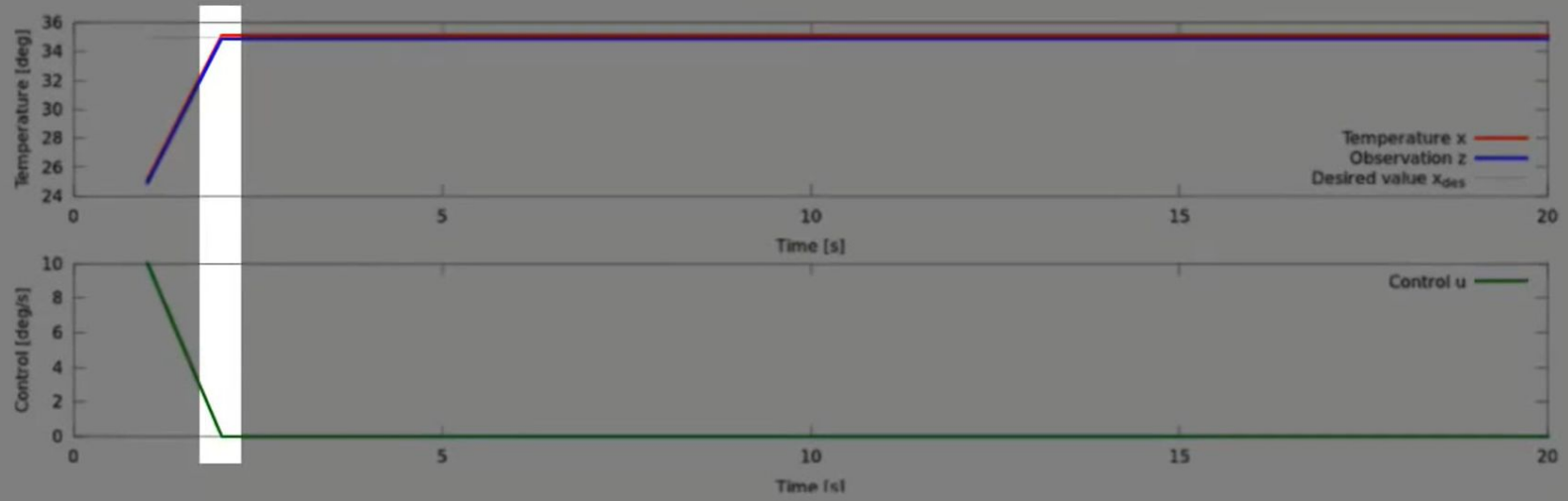
The first step: The algorithm would compare the observed temperature 25° to the desired temperature 35° and find that there is a difference of +10°. So it would feedback the control or it would generate a control of +10°. And within one time step, this control will change the temperature.



Proportional Control

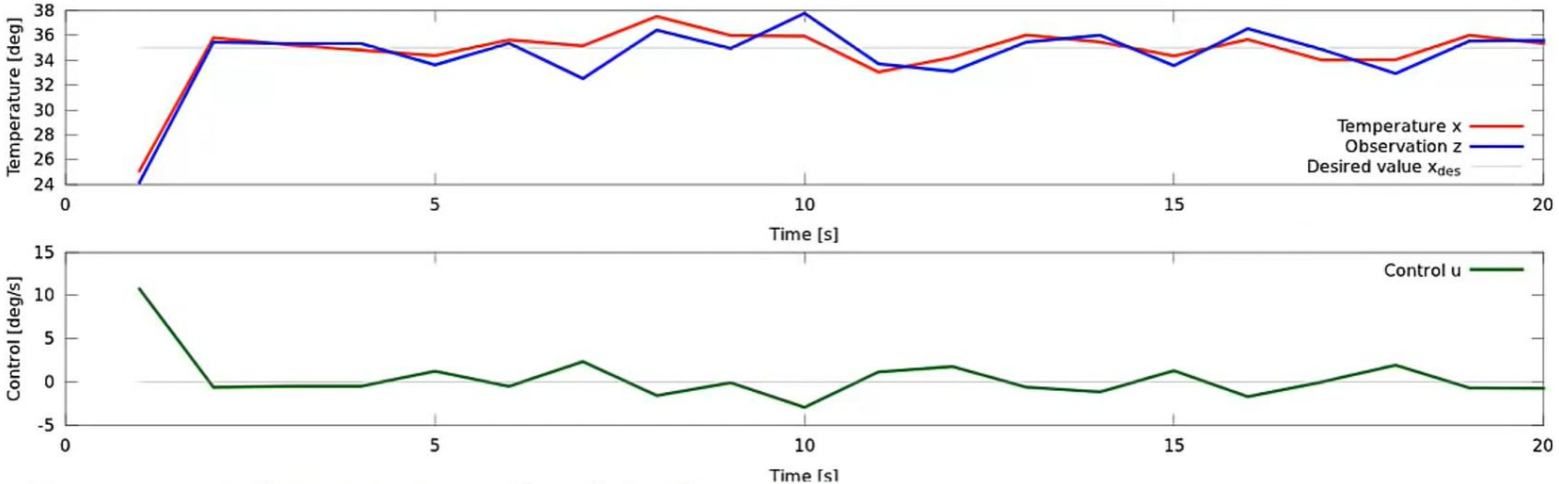
- P-Control: $\mathbf{u}_t = K\mathbf{e}_t$

The next step: we have already reached the 35°, and because we're at the desired value, the error term will become 0 and as a result, the control will become 0.



Effect of Noise

- What effect has noise in the process/measurements?

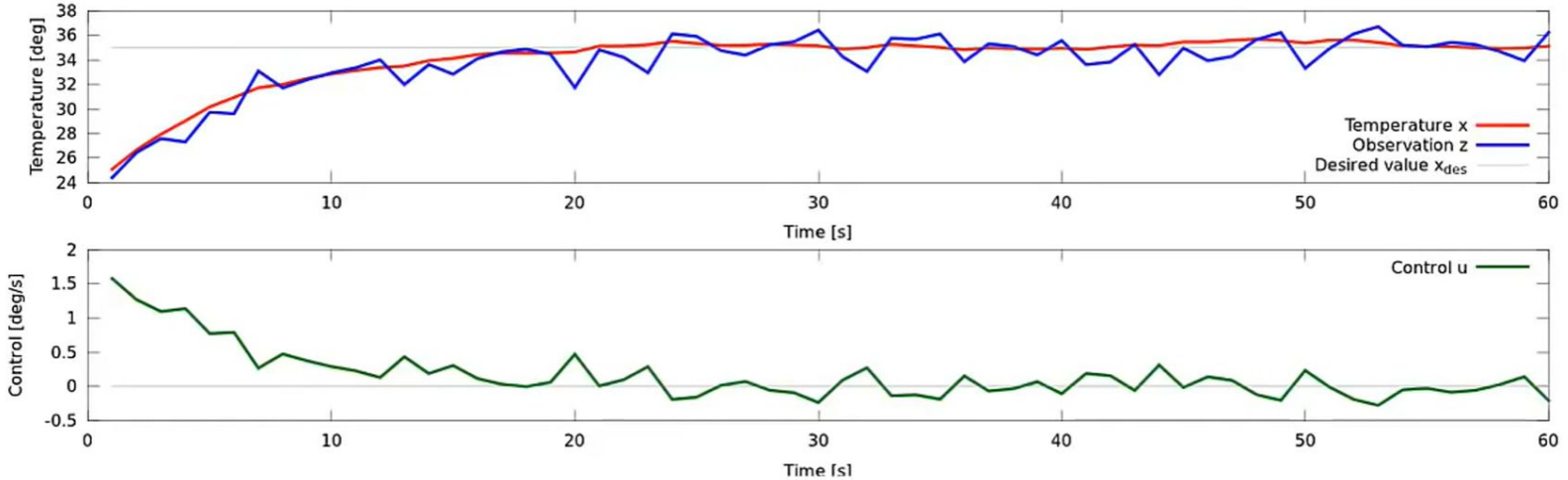


- Poor performance for K=1
- How can we fix this? By reducing the value of K

Proper Control with Noise

- Lower the gain… (K=0.15)

It means that if we observe 10° difference between our current value and the desired one, then we will only try to adjust it by 1.5 degree per time step. This will provide a robust system against noise. But, it will take much larger time to reach the desired value (temperature.)

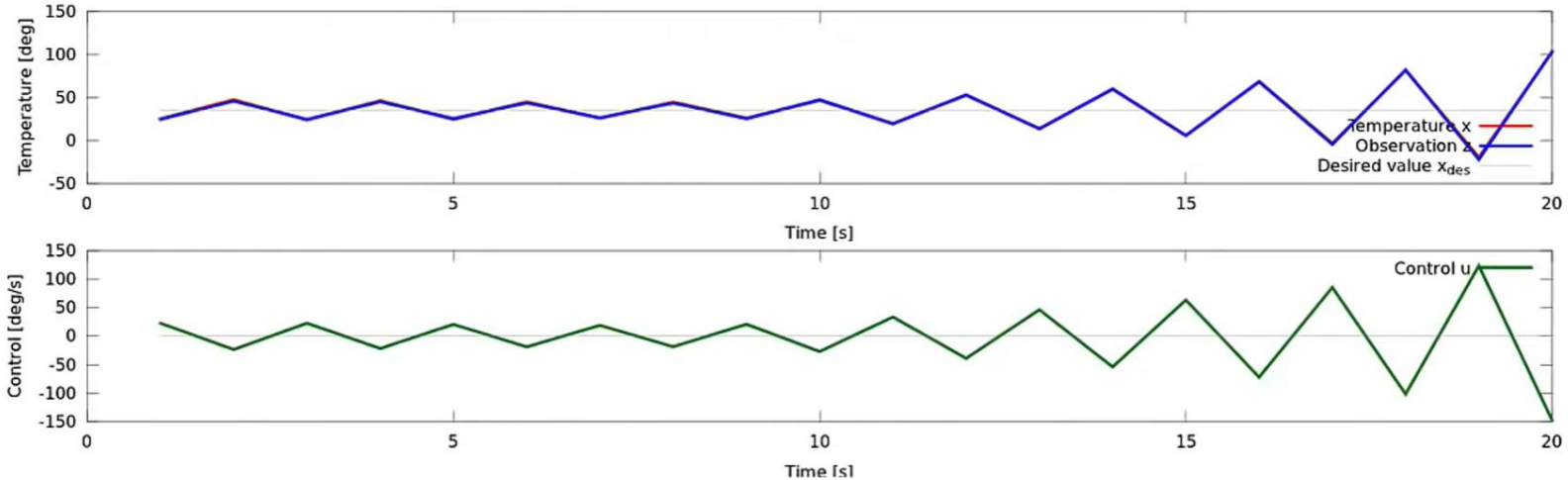


The ***problem*** here is that if you lower the gain (k) too much, the system or the controller will get slower and slower and it will take more time to reach the desired value.

What do High Gains do?

Not a good choice

- High gains are always problematic (K=2.15)

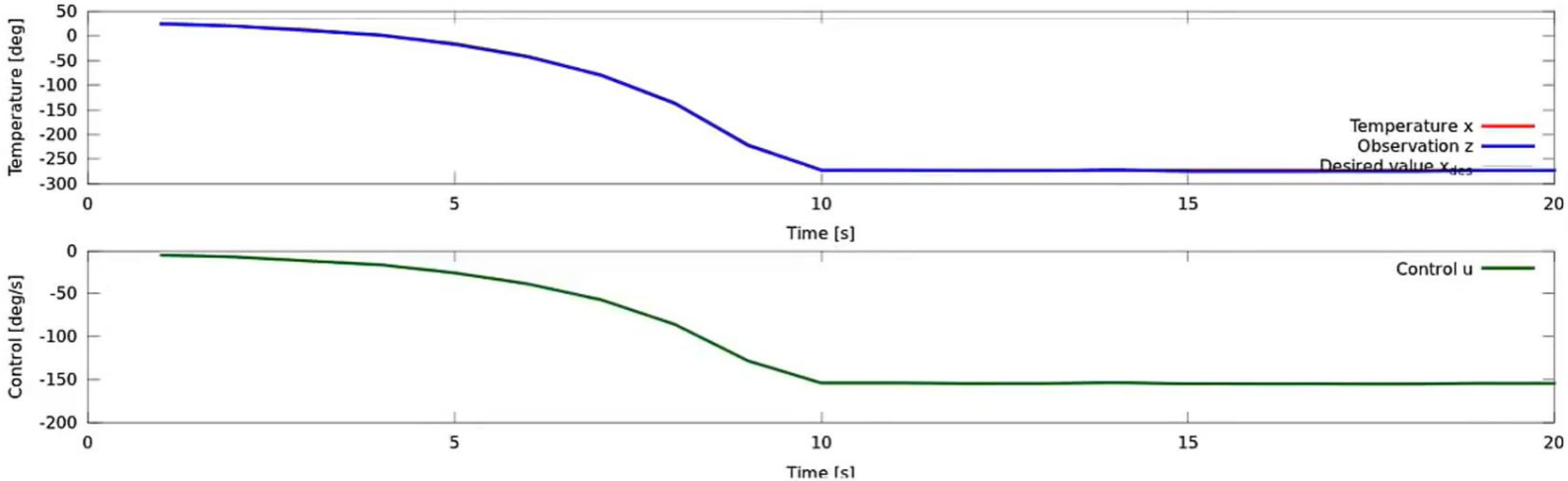


It will overshoot a lot because it finds it's a 10° difference between the current value and the desired value. So, it will multiply that by 2.15 and then add 20° to that and this value will be too high.

What happens if sign is messed up?

- Check K=-0.5

The controller will make the temperature lower than before, and so the temperature will be reduced more and more ⇒ May reach -272° which is unreliable value for application like shower temp. controlling.

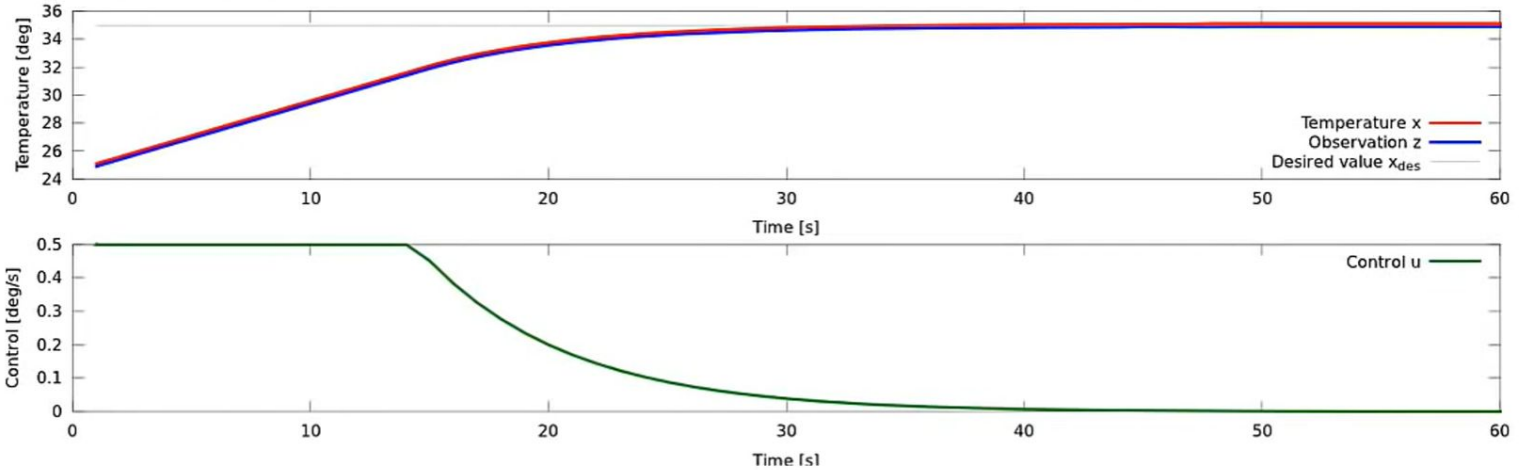


Saturation

Control saturation ⇒ The controller has a maximum and maximum values that you can give.

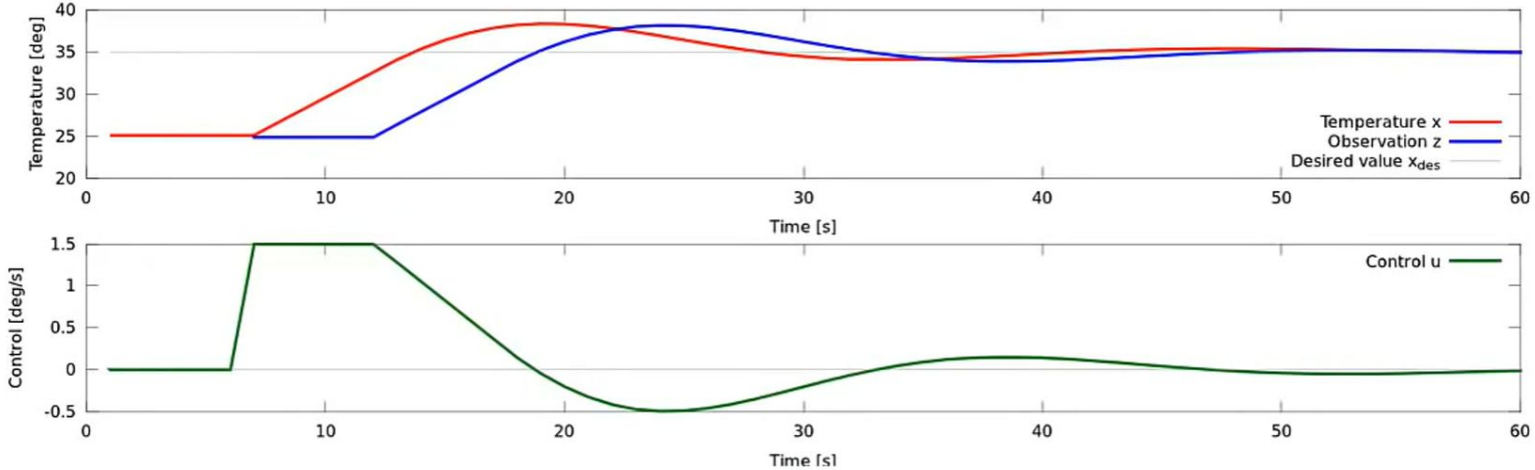
For example, we can control that the shower can't get colder than 0° and can't be hotter than 100° ⇒ This is the idea of saturation ⇒ This exists in the state space

- In practice, often the set of admissible controls u is bounded
- This is called (control) saturation



Delays

- In practice most systems have delays
- Can lead to overshoots/oscillations/de-stabilization

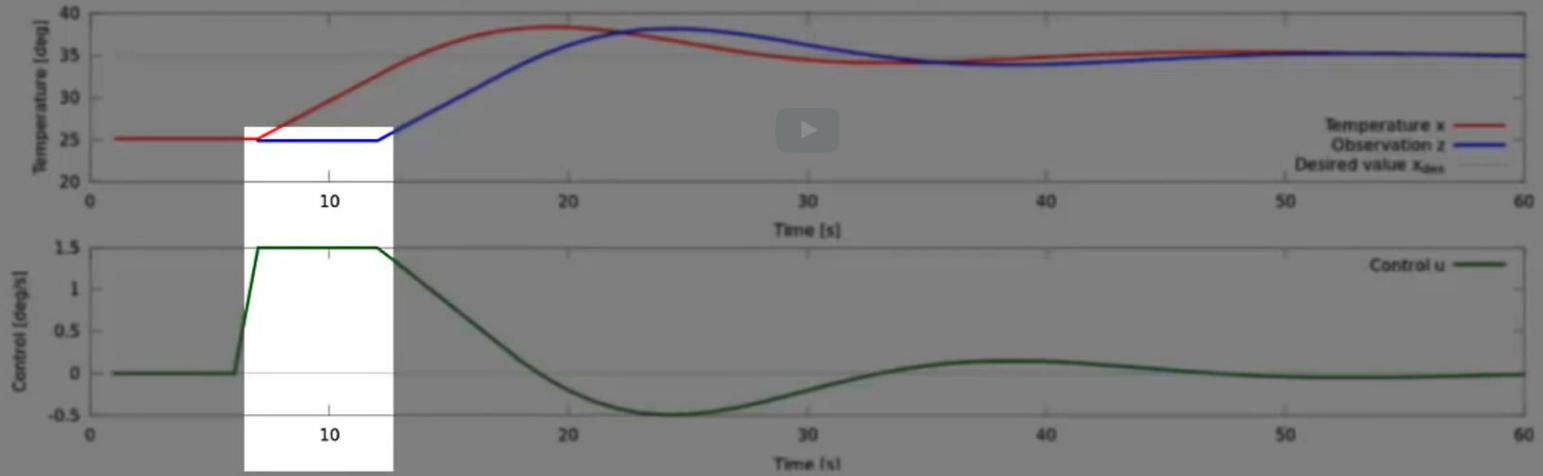


- One solution: lower gains (why is this bad?)

Delays

For the shower example: In time step 8, the valve of the shower is opened or the temperature is being changed, but in the blue line, which gives the observed temperature from our systems, nothing changes for 5 seconds! ⇒ because we have a time delay of 5 seconds..

- In practice most systems have delays
- Can lead to overshoots/oscillations/de-stabilization



- One solution: lower gains (why is this bad?)

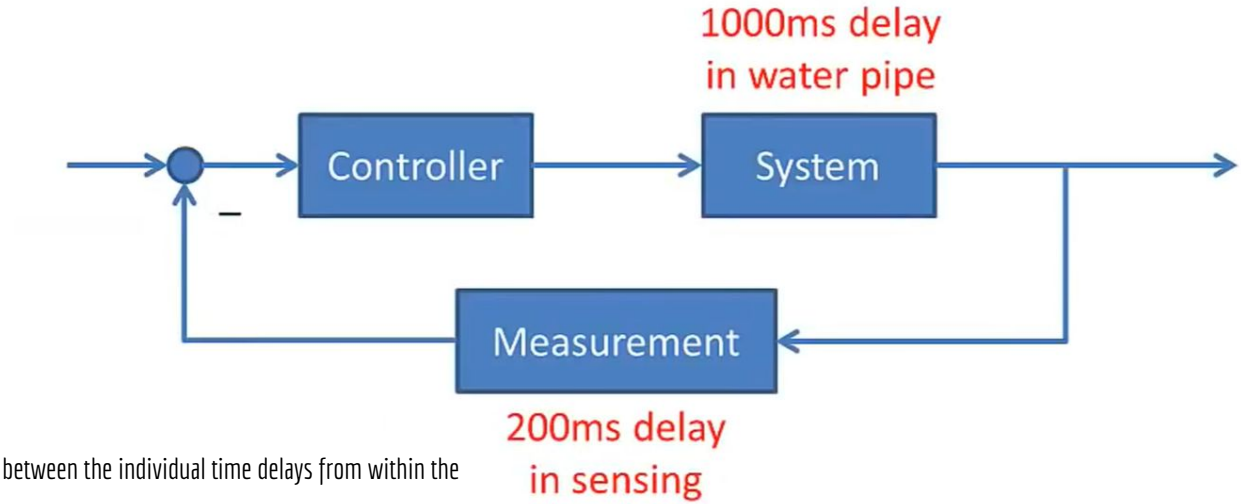
To be continued to the previous slide

This case means that when the observed temperature of the controller reaches the 35°, the temperature in real life will be hotter (››35°) which means that we have to reduce it by passing negative coefficient (-ve K).

One solution for this problem is to pass lower gains ⇒ This is a bas option ⇒ As the convergence takes much time to reach the desired value.

Delays

- What is the total dead time of this system?



Can we distinguish between the individual time delays from within the controller?

- Can we distinguish delays in the measurement from delays in actuation?

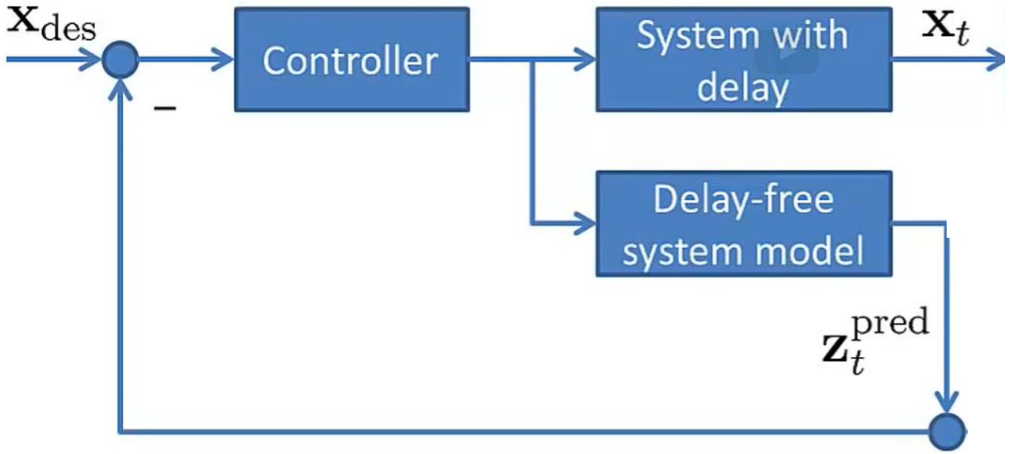
The answer is no, the dead time (delays) of the system is not observable.

Smith Predictor

Another solution is using the smith predictors which allows you to use higher gains , then you could use it if you just ignore the delays.

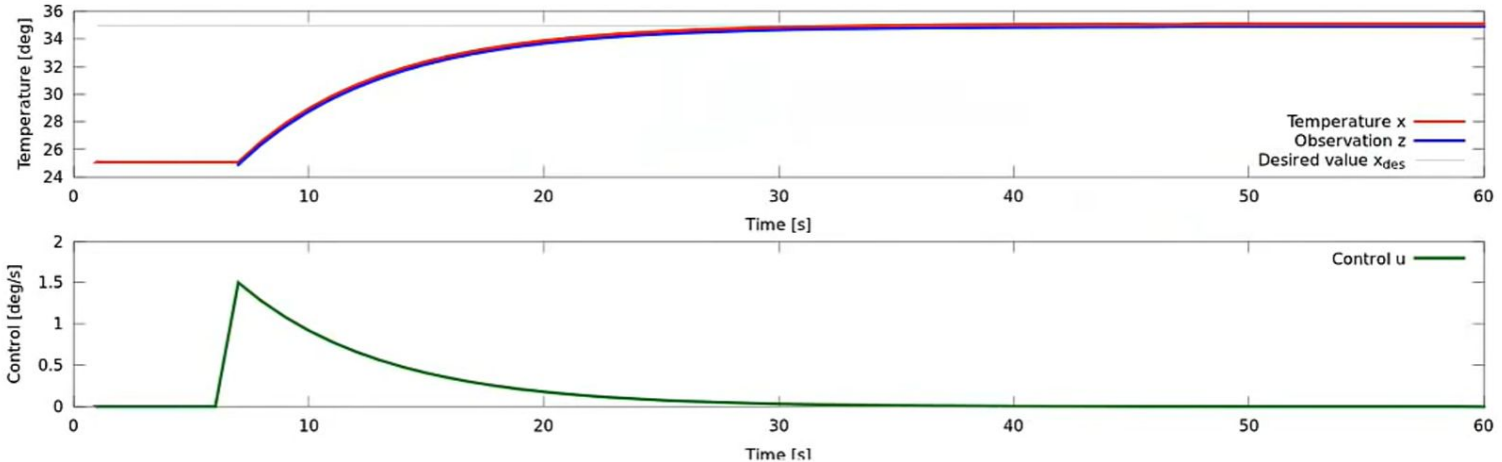
The idea here is taking the observed values directly from the system and back it to the controller again as a feedback without taking any noise or delays under consideration.

- Allows for higher gains
- Requires (accurate) system model

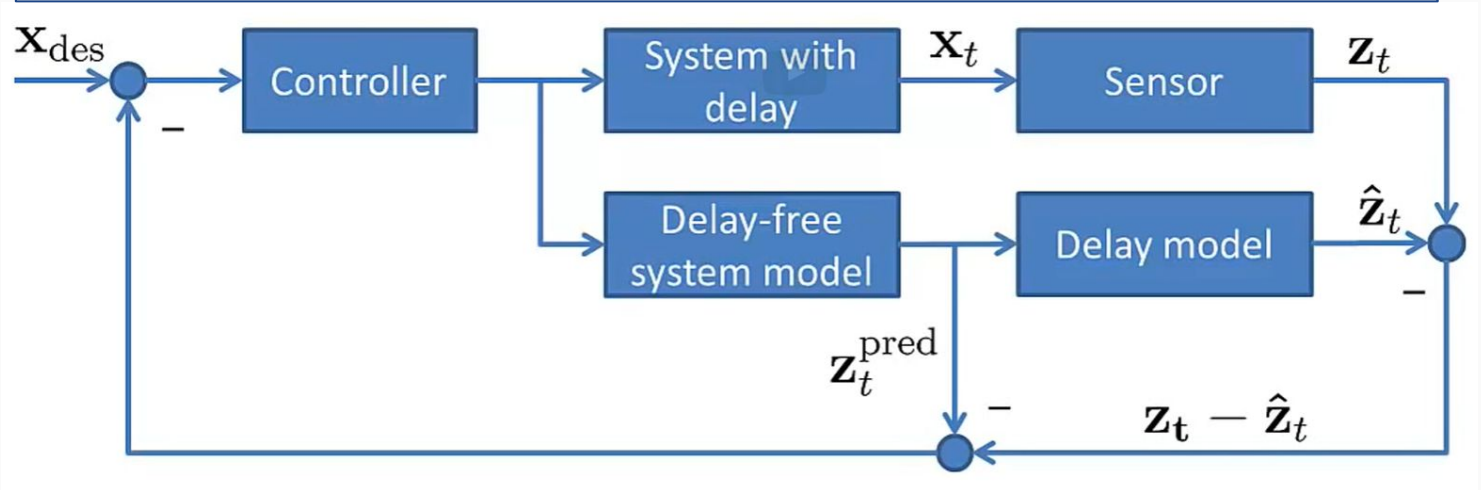


Smith Predictor

- Assumption: System model is available, 5 seconds delay
- Smith predictor results in perfect compensation
- Why is this unrealistic in practice? This would only work if we have a perfectly accurate model, which is never the real situation.



The idea of smith predictors



We have a sensor that observes the system state, but observation (Z_t) is actually time delayed by a certain time delay. But what we do now is also time delay our prediction (Z^_pred) that we get from our system and we compare these two values. And if our model is perfectly accurate, then the difference between the real sensor observation and the delayed sensor observation should be zero.
If it's not perfect, then we will get here a residual that we can use to correct put delay free prediction. And in this way, we can take the advantage of both the delay free system model, but we still can deal with deviations from that. If system model doesn't do anything or is partially wrong, then we still have this normal error term that just considers the difference of sensor and the model.

⇒ You don't know exactly about the delay, in some systems, you know, but if it is a communication delay for example, that you get through wireless, then the delay or the bandwidth might change as you fly and it's not always possible to know how long the signal was travelling.

The question here is whether the better is to overestimate or underestimate the time delay??

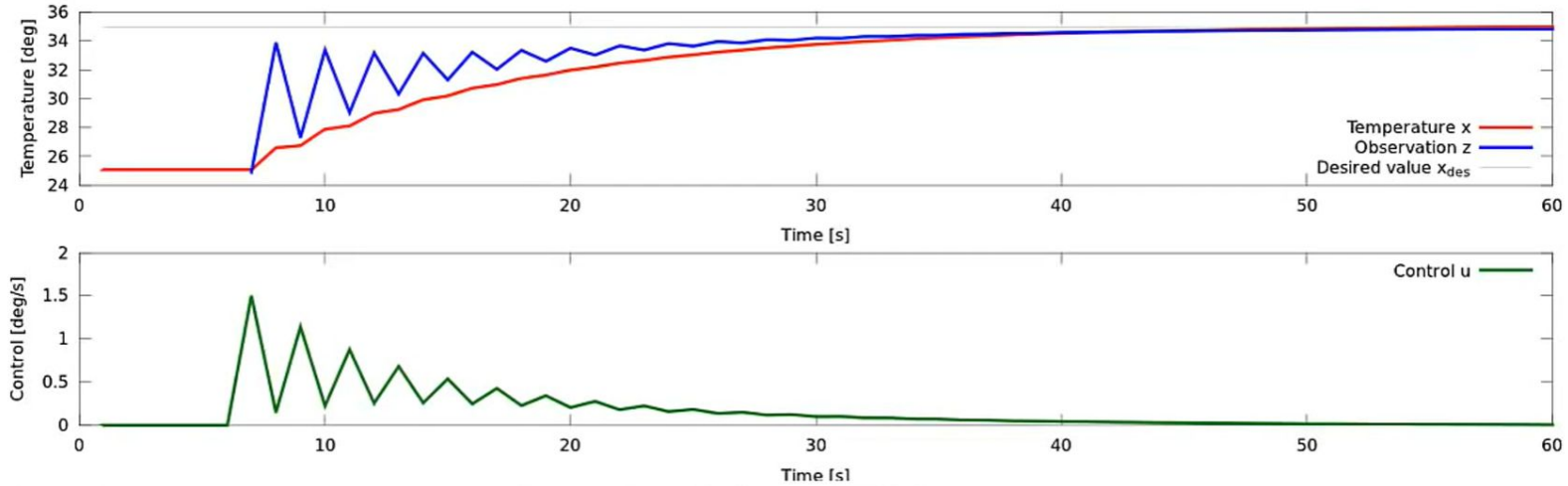
Smith Predictor

- Time delay (and system model) is often not known accurately (or changes over time)
- What happens if time delay is **over**estimated?

The smith predictor predicts so far into the future, which lead to overshooting.
This may reduce the controls in the next time step; because it thinks we already going too fast. It costs a lot of energy as it causes system oscillation.

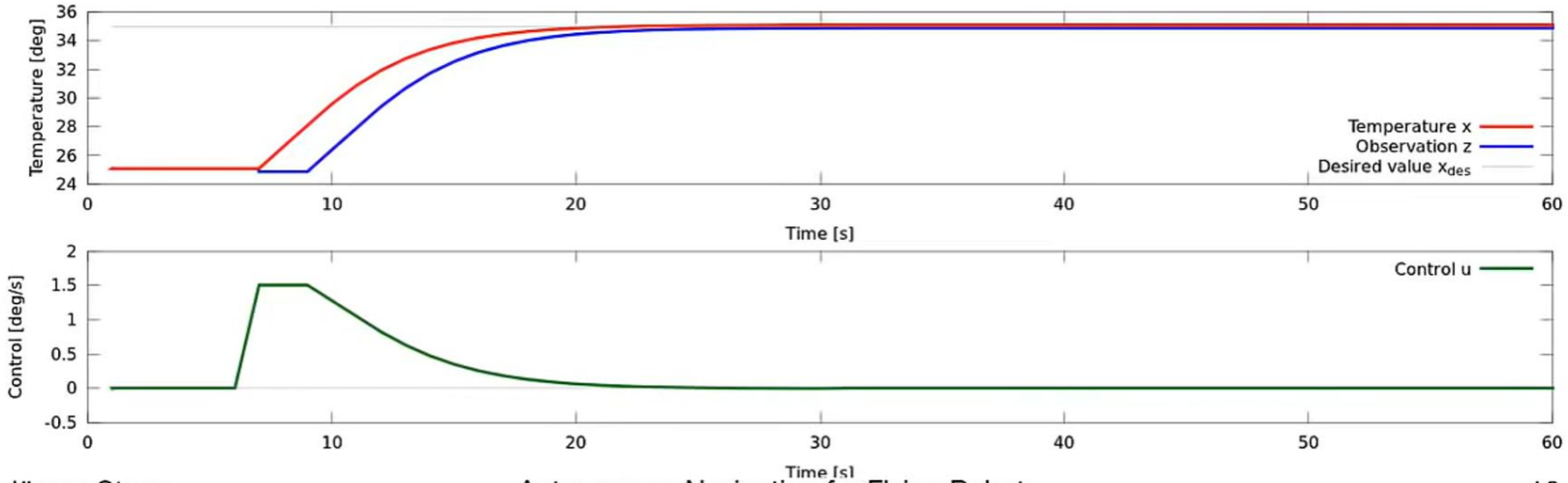
Smith Predictor

- Time delay (and system model) is often not known accurately (or changes over time)
- What happens if time delay is **over**estimated?



Smith Predictor

- Time delay (and plant model) is often not known accurately (or changes over time)
- What happens if time delay is **under**estimated?



In case of underestimating

This case is often done in practice as it doesn't get this overshooting effect of our prediction; because this have a certain discrepancy (error) between the model and the actual system.

In this way, we can compensate for half or may be go perfect of the 90% of the time delay and we consider the rest as the noise of the thing we try to not model, just use directly the sensor readings.

⇒ This leads to a slightly slower convergence in the end but it circumvents oscillations.

Kinematics and Dynamics

How the rigid bodies behave under the influence of forces and torques that are being applied to them.

Kinematics

Describes the motion of the rigid bodies

- Describes the motion of rigid bodies
- Position
- Velocity
- Acceleration

Remember that:

The velocity is the integration of the acceleration.
The position is the integration of the velocity.

Example: 1D Kinematics

- State $\mathbf{x} = \underbrace{\begin{pmatrix} x & \dot{x} & \ddot{x} \end{pmatrix}^\top}_{\text{(Position velocity acceleration)}} \in \mathbb{R}^3$
- Action $u \in \mathbb{R}$
- Time constant $\Delta t \in \mathbb{R}$ ⇒ Tells us how much time passes from one moment to the next.
- Linear process model ⇒ Tells us how we can evolve our state from time step (t-1) to the time step (t).

Control input; means to add the action to the acceleration

$$\mathbf{x}_t = \begin{pmatrix} 1 & \Delta t & 0 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{pmatrix} \mathbf{x}_{t-1} + \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} u_t$$

From that, we can get a simple dynamical system of an object that floats in 1D space.

- *The first row of the matrix:*

Current position = previous position + delta_t * current speed

- *The second row:*

Current speed depends on the previous speed + delta_t * current acceleration

- *Last row:*

Keep the acceleration the same as long as nothing happens.

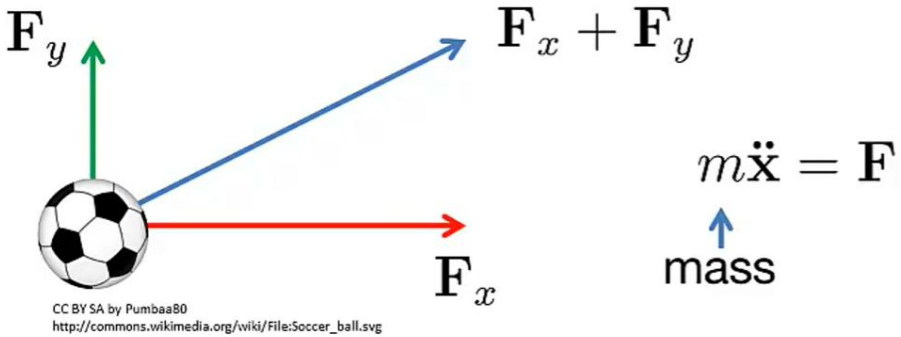
Dynamics

- Actuators induce forces and torques
- Forces induce linear acceleration
- Torques induce angular acceleration

Torque ⇒ is the turning-force that makes something turn.

Forces and Accelerations

- Forces are vectors and can be summed up
- Important forces (for us): Gravity, thrust, friction
- Forces induce accelerations



$$m\ddot{\mathbf{x}} = \mathbf{F}$$

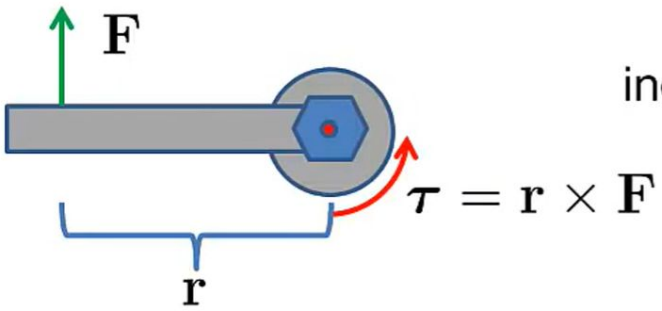
mass

Torques and Angular Accelerations

- Force on a lever induces a torque (“turning force”)
- Forces are vectors and can be summed up
- Torque results in angular acceleration α

$$J\alpha = \tau$$

inertial matrix



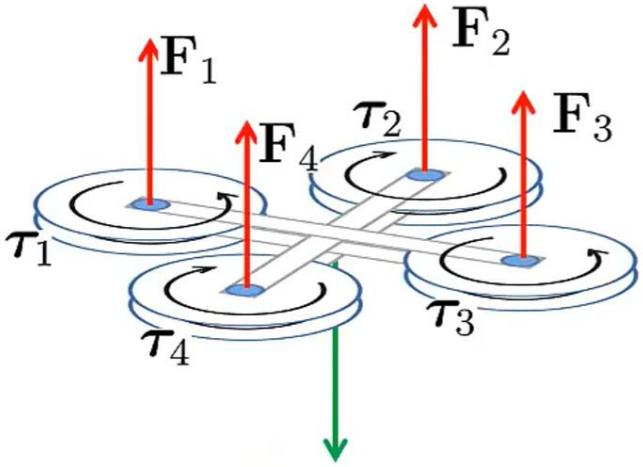
$$\tau = \mathbf{r} \times \mathbf{F}$$

If you have a lever-arm that you try to move, you can apply a force on one side and that will induce then a torque on the rotation axis.

It depends on the mass or the mass distribution on the object, which is typically captured by J (inertial matrix) that related to the torque to the angular acceleration α ($J\alpha = \mathcal{T}$)

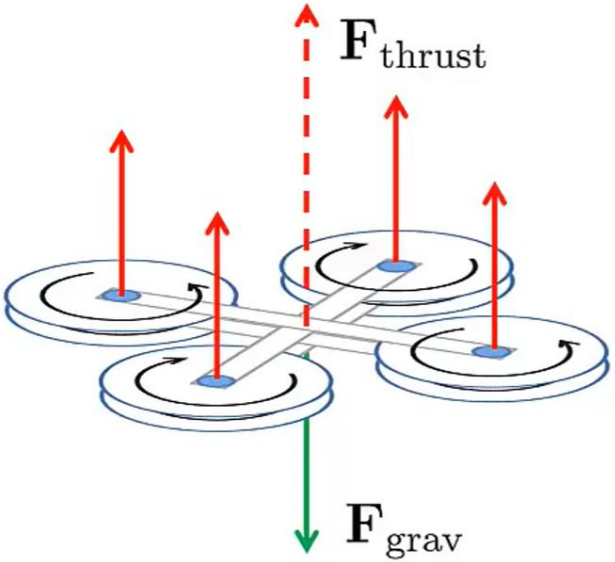
Dynamics of a Quadrotor

- Each propeller induces force and torque by accelerating air
- Gravity pulls quadrocopter downwards



Vertical Acceleration

- Thrust $\mathbf{F}_{\text{thrust}} = \mathbf{F}_1 + \mathbf{F}_2 + \mathbf{F}_3 + \mathbf{F}_4$



Explanation for slides 41 & 42

⇒ The quadcopter consists of 4 propellers, every propeller generates some thrust, and this induces a certain force. So, we have now 4 sources of force that push the quadrotor up.

In addition, we have the gravitational force that pulls the quadrotor down at the same time.

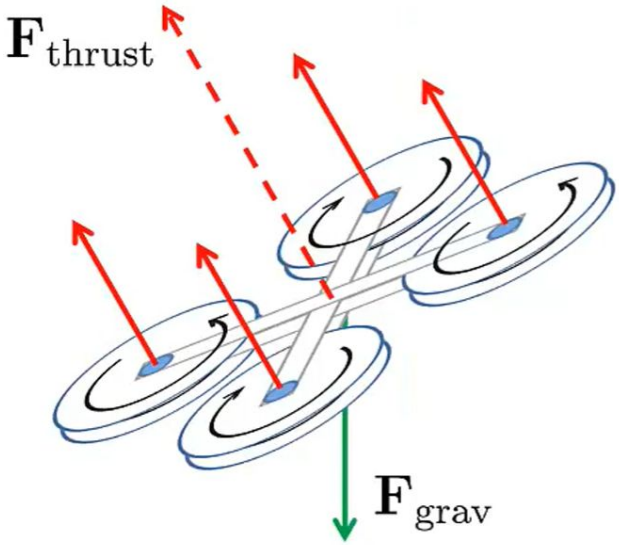
⇒ If we count all of these forces together in the vertical case, then the quadrotor hovers exactly in place, if the thrust forces of all factors match the gravitational force that pulls it down again.

⇒ Simultaneously, every propeller also induces the torque, which is the counter torque that's generated by the propellers moving the air.

⇒ **We have to make sure that all of this torques actually sum to 0, in order not to get any motion.**

Vertical and Horizontal Acceleration

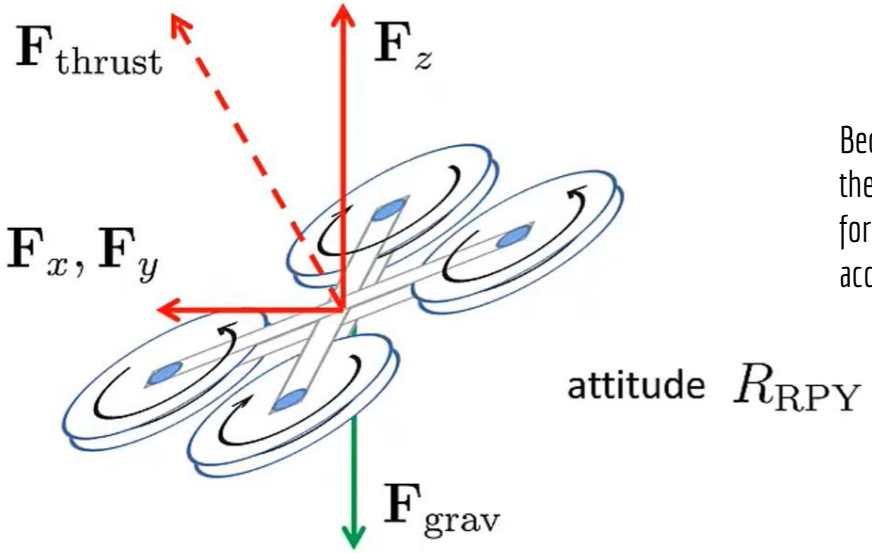
- Thrust $\mathbf{F}_{\text{thrust}} = \mathbf{F}_1 + \mathbf{F}_2 + \mathbf{F}_3 + \mathbf{F}_4$



If we turn the quadrotor slightly, then this thrust force is not in a line any more with the gravitational, this means that we can separate this resulting force from the motor thrust into a vertical component F_z that compensates for the gravity if you want to keep the same position. And an x & y component that actually makes the quadrotor go forward or sideways.

Vertical and Horizontal Acceleration

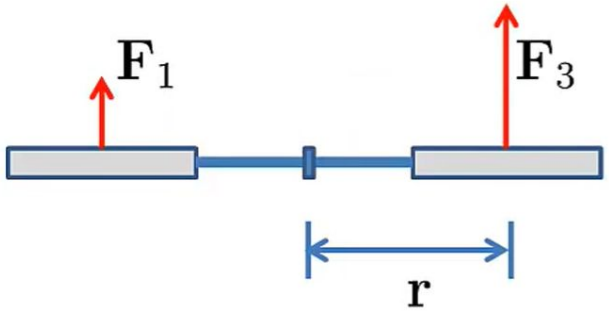
- Thrust $\mathbf{F}_{\text{thrust}} = \mathbf{F}_1 + \mathbf{F}_2 + \mathbf{F}_3 + \mathbf{F}_4$
- Acceleration $\ddot{\mathbf{x}}_{\text{global}} = (R_{RPY}\mathbf{F}_{\text{thrust}} - \mathbf{F}_{\text{grav}})/m$



Because all of those forces are vectors, we have to transform the thrust vector into the global coordinate system by using for example a rotation matrix to be able to compute the acceleration in the global coordinates.

Pitch (and Roll)

- Attitude changes when opposite motors generate unequal thrust
- Induced torque $\boldsymbol{\tau} = (F_1 - F_3) \times \mathbf{r}$
- Induced angular acceleration $\boldsymbol{\alpha} = J^{-1}\boldsymbol{\tau}$

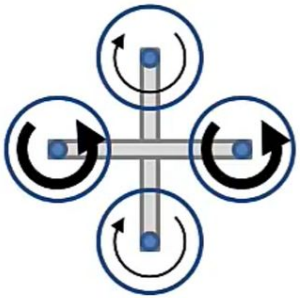


Side view of quadrotor

If we want to change the attitude of the quadrotor, we can bring the motors into a mode where the two opposite motors produce an unequal thrust, and this then induces an certain torque around the rotation axis or the center of the mass of the quadrotor.

Yaw

- Each propeller induces torque due to rotation and the interaction with the air
- Induced torque
- Induced angular acceleration $\tau = \tau_1 - \tau_2 + \tau_3 - \tau_4$

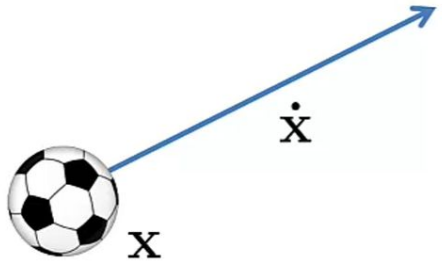


For rotating around the vertical axis, we can also modulate the speeds of the rotors in such a way that the sum of the torque does not equals 0 anymore, but the left turning motors are turning faster than the right turning ones for example. That induces the torque on the overall system.

PID Control

Rigid Body Kinematics

- Consider a rigid body
- Free floating in 1D space, no gravity

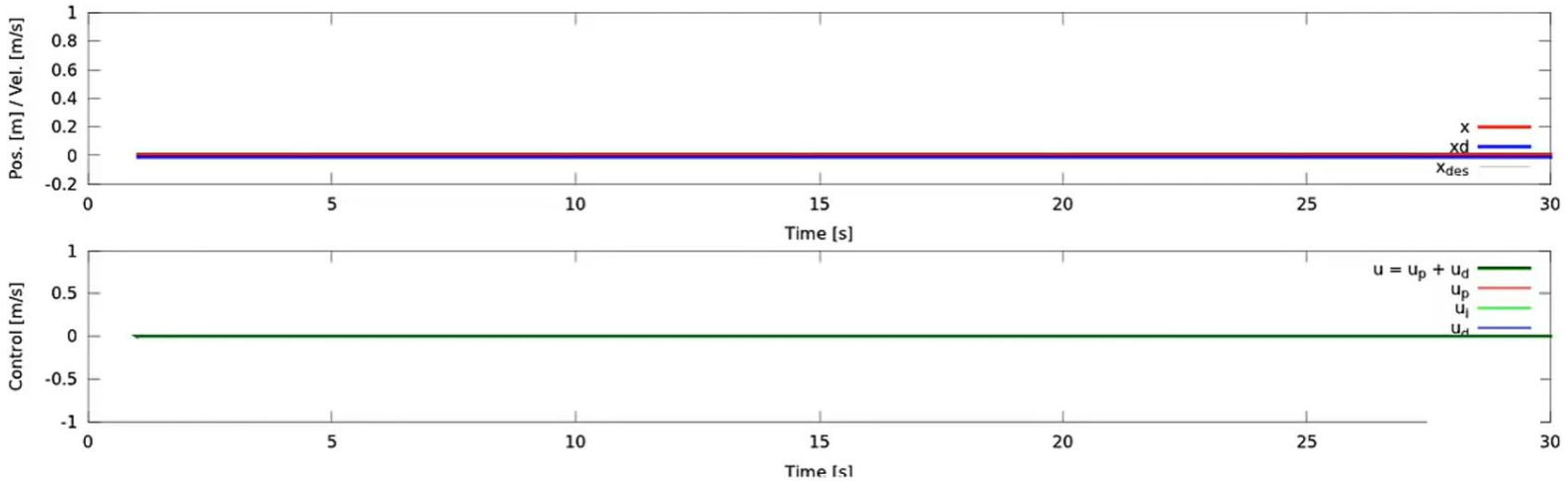


Rigid Body Kinematics

Current position = previous position + current speed delta_t = 1, time constant

- System model $\mathbf{x}_t = \mathbf{x}_{t-1} + \dot{\mathbf{x}}$
- Initial state $\mathbf{x}_0 = 0, \dot{\mathbf{x}}_0 = 0$

Here, we can see that nothing changes because in every new time step, we just add 0 + 0 and that means that our object will stay exactly at the same point.



Rigid Body Kinematics

- In each time instant, we can apply a force $\mathbf{F}_t \propto \mathbf{u}_t$
- Results in acceleration $\ddot{\mathbf{x}}_t = \mathbf{F}_t / m$
- Desired position $\mathbf{x}_{\text{des}} = 1$

- What will happen if we apply P-control?

$$\mathbf{u}_t = K(\mathbf{x}_{\text{des}} - \mathbf{x}_{t-1})$$

U_t ⇒ control signal
K ⇒ control gain
X_des ⇒ desired location
X_t-1 ⇒ Current location from the previous time step

P Control

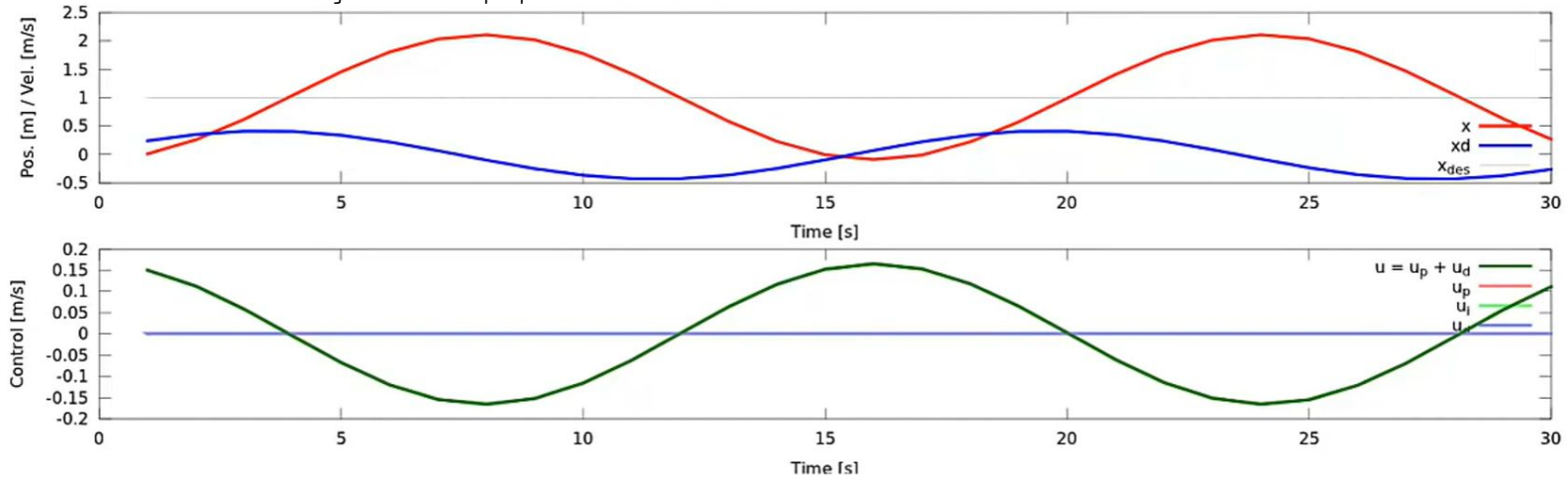
What would happen if we apply [u_t = k(x_des - x_t-1)] to a grid body?

- Control law

$$\mathbf{u}_t = K(\mathbf{x}_{\text{des}} - \mathbf{x}_{t-1})$$

The P-Controller will generate a large control in this case of 0.15 as the k = 0.15, which will make the object accelerates towards its desired location.

We start at 0 and want to go to a location of 1 if k = 0.15



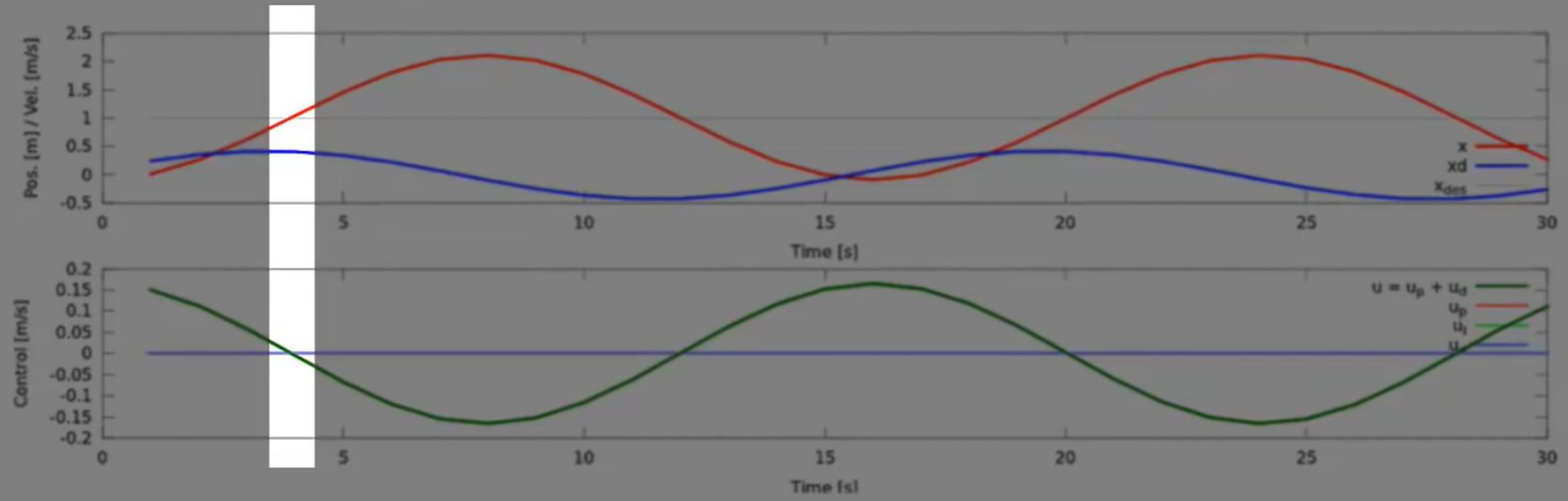
P Control

- Control law

$$\mathbf{u}_t = K(\mathbf{x}_{\text{des}} - \mathbf{x}_{t-1})$$

At 4 seconds, we will hit the desired location of 1, but because we still have some inertia, the object will continue beyond this point and overshoot massively up to a position of 2.
During this time, the controller will generate a negative control output to decelerate the object. This leads to a very bad oscillation. This will not make our object coma closer to the desired location.

This is not good, P-Controls of rigid bodies that are not damped are not sufficient.



PD Control

The idea is to use a derivative term

- Proportional-derivative control

$$\mathbf{u}_t = K_P(\mathbf{x}_{\text{des}} - \mathbf{x}_{t-1}) + K_D(\dot{\mathbf{x}}_{\text{des}} - \dot{\mathbf{x}}_{t-1})$$

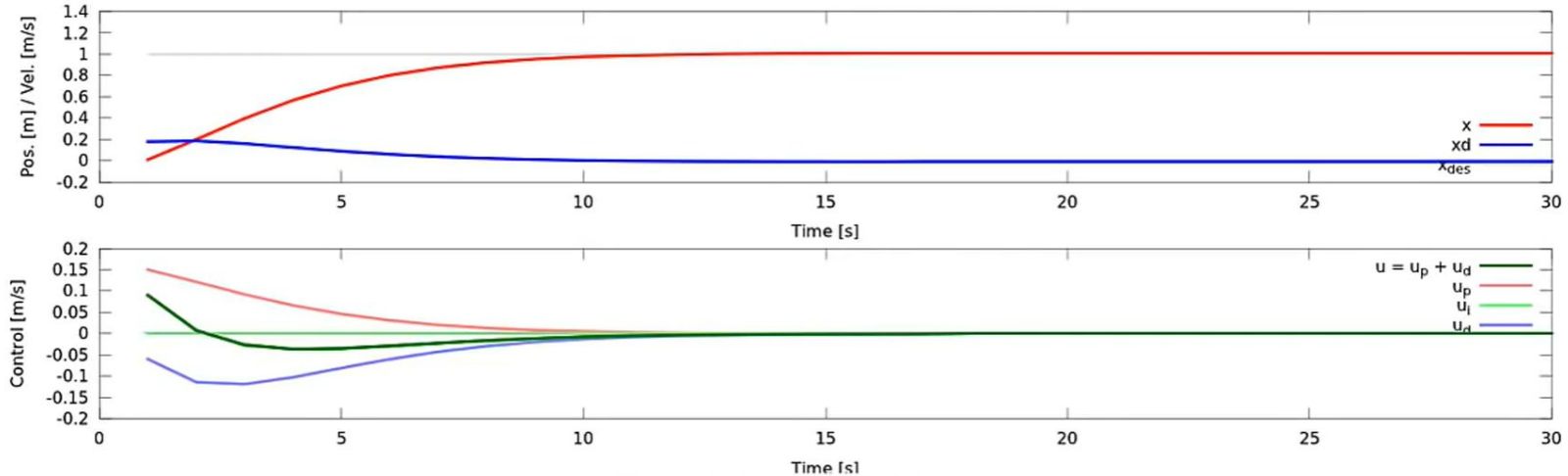
Derivative term for minimizing the distance between the desired speed that we try to achieve and our current speed.

PD Control

- Proportional-derivative control

$$\mathbf{u}_t = K_P(\mathbf{x}_{\text{des}} - \mathbf{x}_{t-1}) + K_D(\dot{\mathbf{x}}_{\text{des}} - \dot{\mathbf{x}}_{t-1})$$

For the same previous example, this means that at the location of 1, we desire to have a speed of 0. So, our $\dot{x} = 0$ to make sure that at the end when we reach the goal, we want to have 0 velocity.



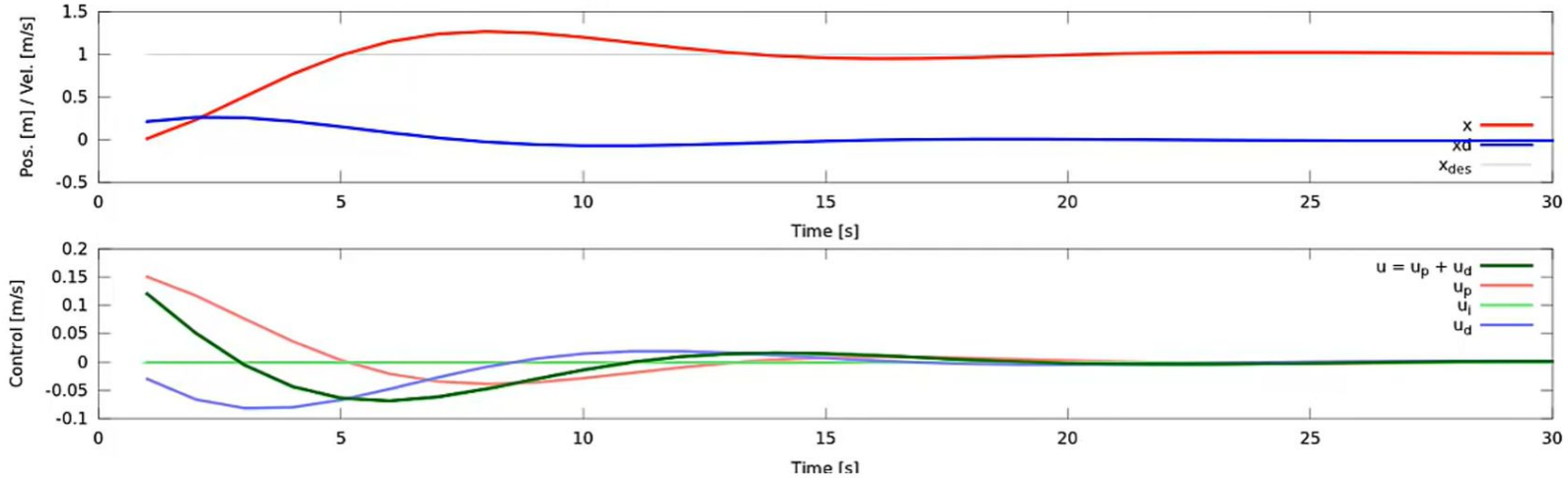
PD Control

- Proportional-derivative control

$$\mathbf{u}_t = K_P(\mathbf{x}_{\text{des}} - \mathbf{x}_{t-1}) + K_D(\dot{\mathbf{x}}_{\text{des}} - \dot{\mathbf{x}}_{t-1})$$

- What if we set lower gains for K_D?

The controller will overshoot to our desired location, because the slowing down is not strong enough.



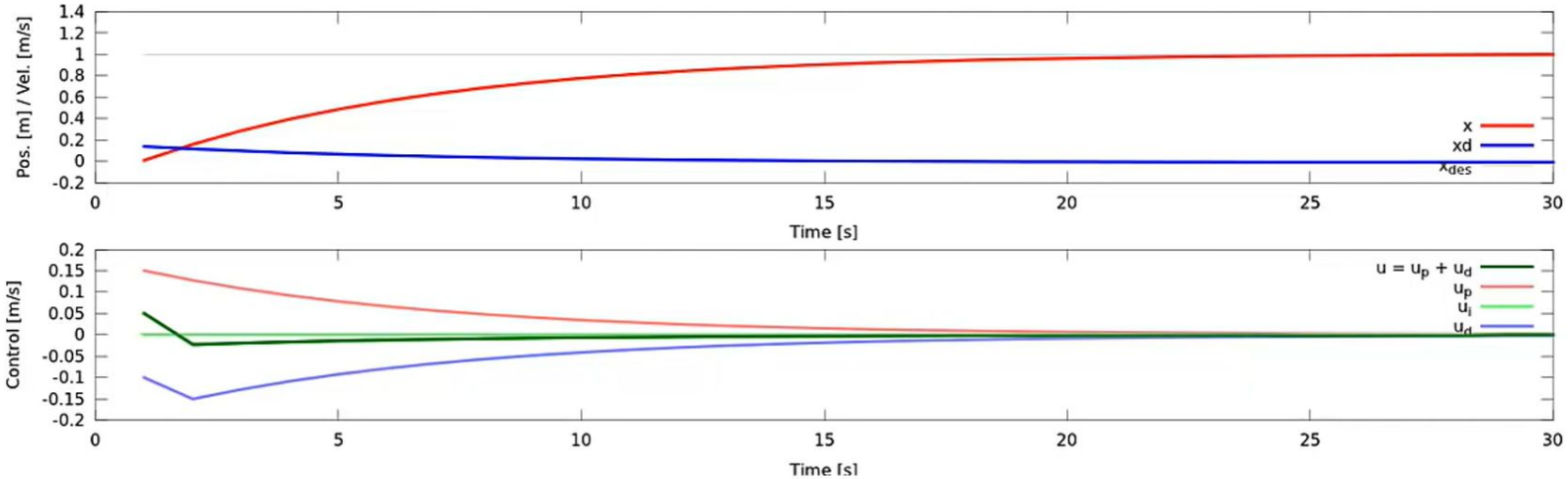
PD Control

- Proportional-derivative control

$$\mathbf{u}_t = K_P(\mathbf{x}_{\text{des}} - \mathbf{x}_{t-1}) + K_D(\dot{\mathbf{x}}_{\text{des}} - \dot{\mathbf{x}}_{t-1})$$

- What if we set higher gains for K_D?

It will still approaches the desired location, but it takes extremely long time like 25 seconds in this example, and it could be much faster. So this K_D also can lead to slower convergence at the end.



PD Control

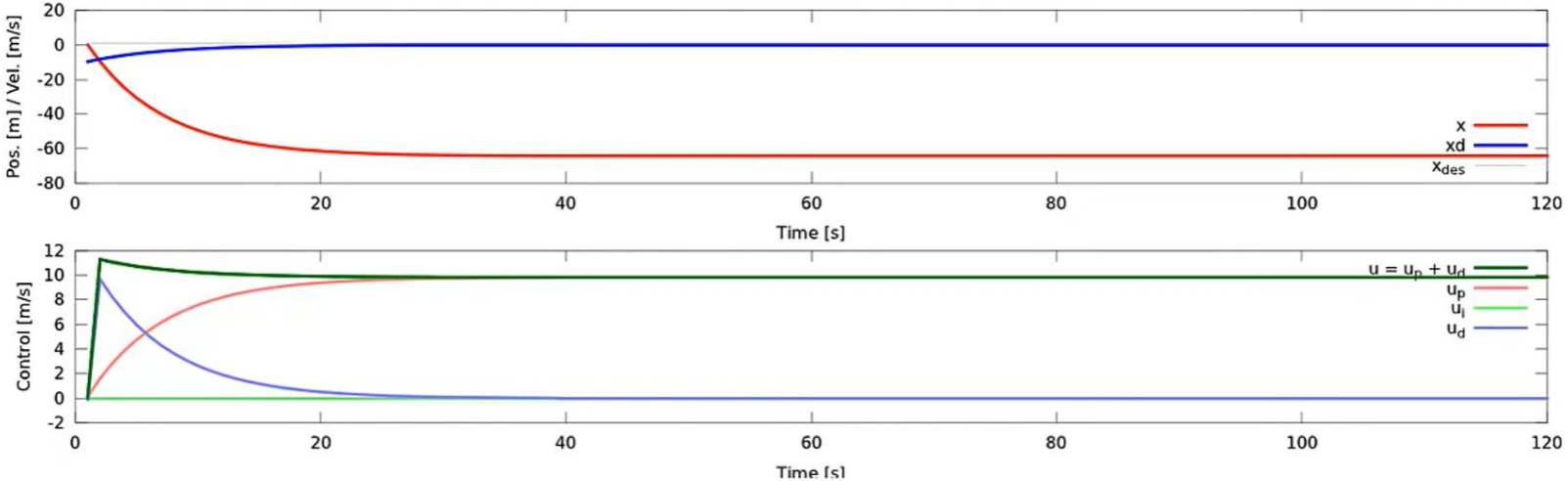
Next to our controller force generated, we add the gravitational constant that accelerates our mass with a certain force.

- What happens when we add gravity?

$$\ddot{\mathbf{x}}_t = (\mathbf{F}_t + \mathbf{F}_{\text{grav}})/m$$

We start at 0, and we want to go to 1, but the object falls down roughly to 60 meters. After about 20 seconds, the PD-Controller generates Sufficiently strong control signals to actually prevent the object falling down. This is not what we want!

This is what happen if we use PD-Control without gravity compensation. The quadrotor will not reach its desired location.



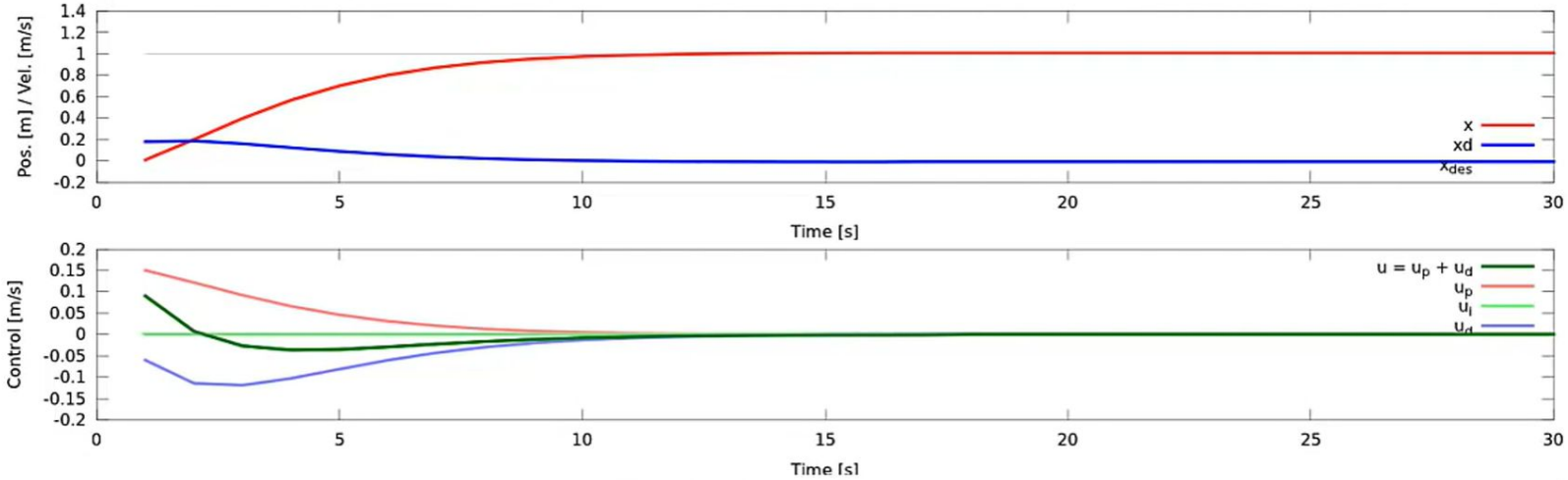
Gravity compensation

The effect is we are going smoothly from 0 to 1

- Add as an additional term in the control law

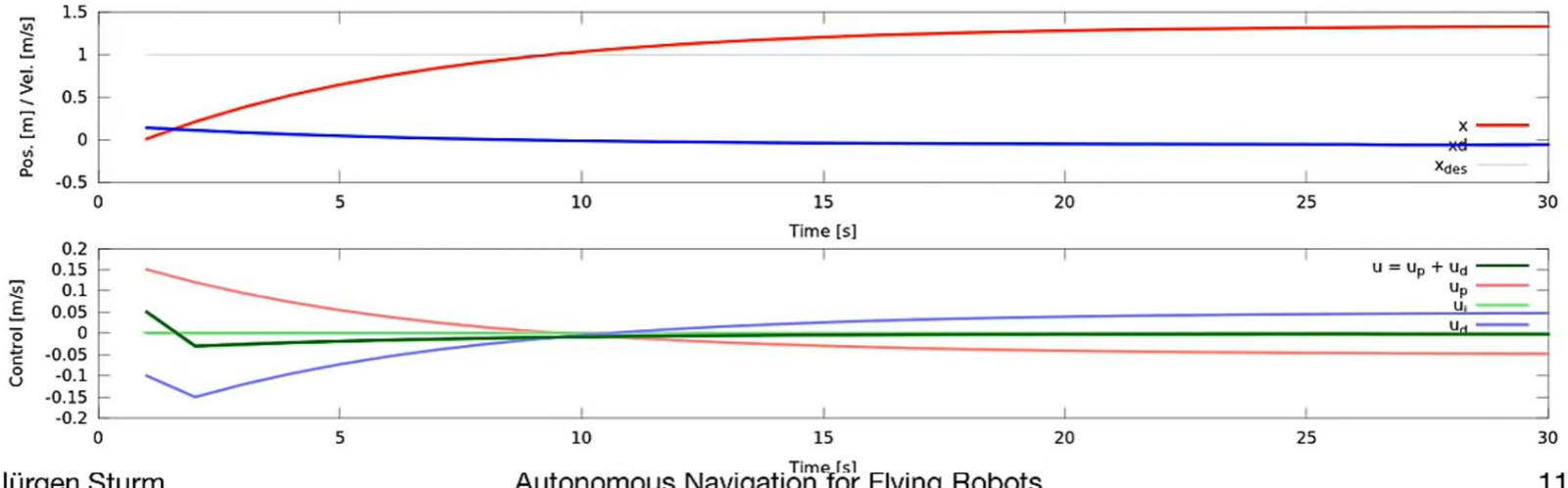
$$\mathbf{u}_t = K_P(\mathbf{x}_{\text{des}} - \mathbf{x}_{t-1}) + K_D(\dot{\mathbf{x}}_{\text{des}} - \dot{\mathbf{x}}_{t-1}) - \mathbf{F}_{\text{grav}}$$

- Any known (inverse) dynamics can be included



PD Control

- What happens when we have systematic errors? (control/sensor noise with non-zero mean)
- Example: unbalanced quadrotor, wind, …



Systematic error

- **Reason of occurrence**
  It could be happen if one motor is a little bit weaker than another one. For example, because it has a broken propeller or just might be a little bit unbalanced on the quadrotor or there might be some external disturbance like wind that constantly pushes a way the quadrotor in a certain direction.

- **Effect**
  If we add something like wind from below that pushed the object upwards (Similar to gravitational force) ⇒ The object will approach the desired a location of 1, but it overshoots and it converges on almost 1.3 may be. And on this location, the PD-controller generates enough control output to not deviate any further from our desired value.

  But at the end, it will not converge on our desired location.

So, the question now is how can we deal with such systematic errors, how can we extend our controller to deal with something like this?

The answer is to estimate this bias term (systematic error) term by integrating overall errors that we had before in the very beginning to the current time step ⇒ **PID Controller**

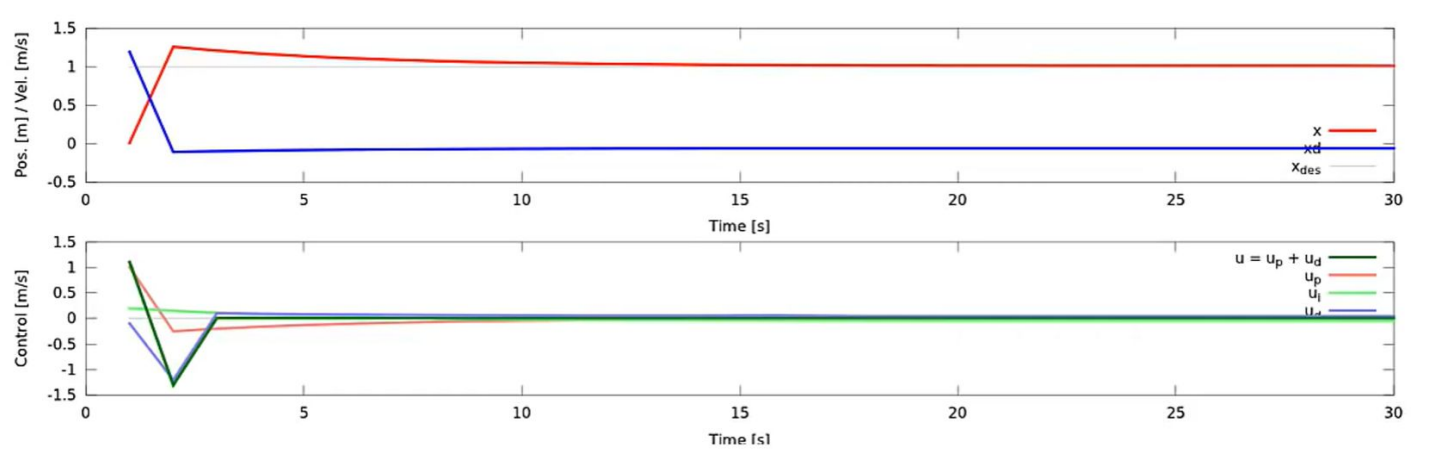
PID Control

Idea: Estimate the system error (bias) by integrating error

$$\mathbf{u}_t = K_P(\mathbf{x}_{\text{des}} - \mathbf{x}_{t-1}) + K_D(\dot{\mathbf{x}}_{\text{des}} - \dot{\mathbf{x}}_{t-1}) + \boxed{K_I \int_0^t \mathbf{x}_{\text{des}} - \mathbf{x}_{t'-1} \mathrm{d}t'}$$

→ Integral term

For the same previous example:



At first, we get some overshooting because initially the integral term is 0, so the algorithm doesn't know that we have a disturbance. Then, the controller will nicely converges on the desired location

In the bottom control plot:

- The red line corresponds to the proportional control output
- The green line corresponds to the differential output.
- The bright green line corresponds to the systematic integral term.

PID Control

- Idea: Estimate the system error (bias) by integrating error

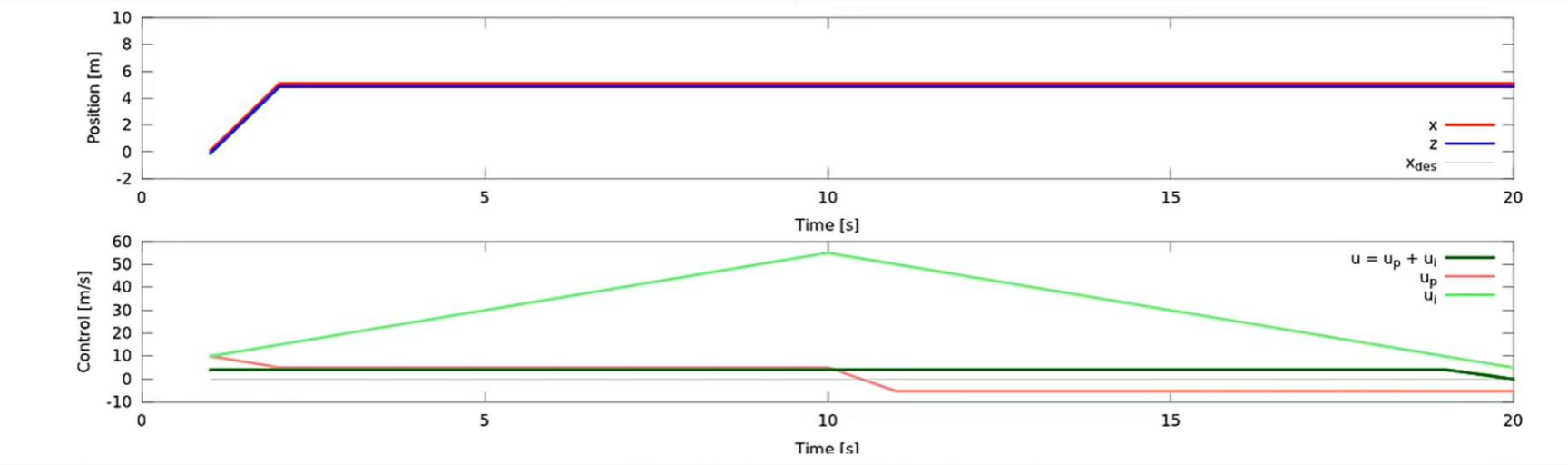
$$\mathbf{u}_t = K_P(\mathbf{x}_{\text{des}} - \mathbf{x}_{t-1}) + K_D(\dot{\mathbf{x}}_{\text{des}} - \dot{\mathbf{x}}_{t-1}) + K_I \int_0^t \mathbf{x}_{\text{des}} - \mathbf{x}_{t'-1} \mathrm{d}t'$$

- For steady state systems, this can be reasonable
- Otherwise, it may create havoc or even disaster (wind-up effect)

Using the integral term would be fine for steady state systems, so systems that can actually reach their desired locations, but we may have systems that can never reach a certain location, and it might create wind-up effect, which can lead to a disaster or at least to a strong delay in our controller.

Example: Wind - up effect

Imagine that we want the quadrotor to go to a certain location, but it on its way get stuck in a tree. So, it can't reach the steady state. Then, we should look at the effect of the being stuck on the integral term.

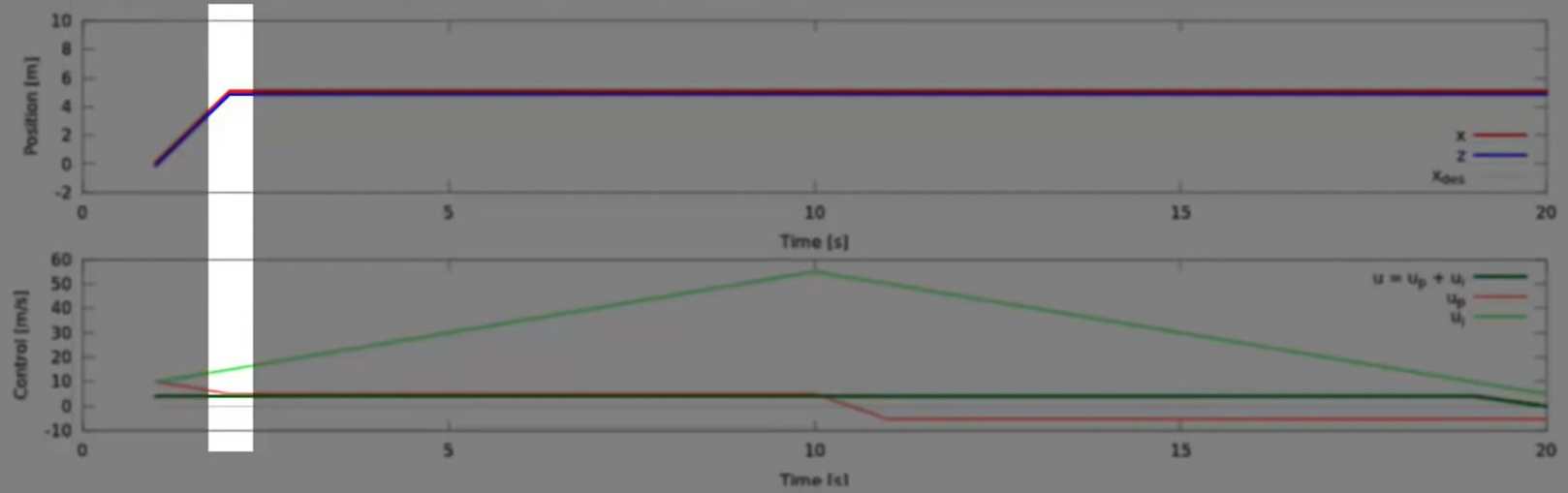


If we willing to send the quadrotor to location of 10, but it got stuck in the tree, the integral term will keep building up; because it realizes that it need to give more power to the motors apparently ti reach the goal location, but it can't; because it is stuck. So, the integral term grows larger and larger as seen in the bright green line in the bottom control plot.

This is why the integral term is sometimes a bit difficult and dangerous to be used.

Example: Wind-up effect

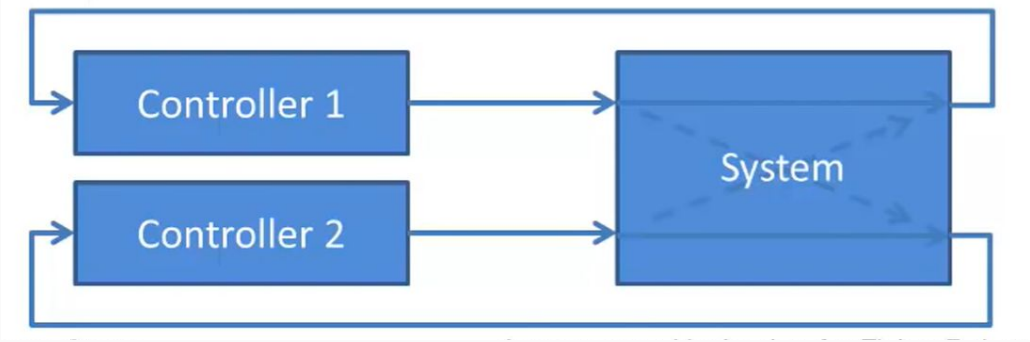
- Quadrotor gets stuck in a tree → does not reach steady state
- What is the effect on the I-term?



Decoupled control

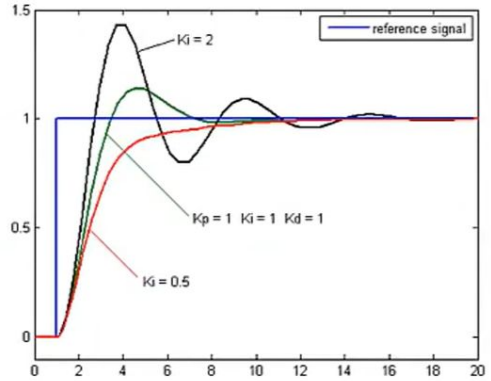
Another problem is that we generally not only have one degree of freedom that we try to control, but typically multiple.
In case of drone, we may have 4 degrees of freedom [go forward, sideward, rotate, changing the flight height]

This is called **MIMO** system [Multiple Input Multiple Output]. Most of these controllers must be decoupled. So, we have 2 or 4 independent PID controllers, each one just controlling one degree of freedom.



How to Choose the Coefficients?

- Gains too large: overshooting, oscillations
- Gains too small: long time to converge
- Heuristic methods exist
- In practice, often tuned manually



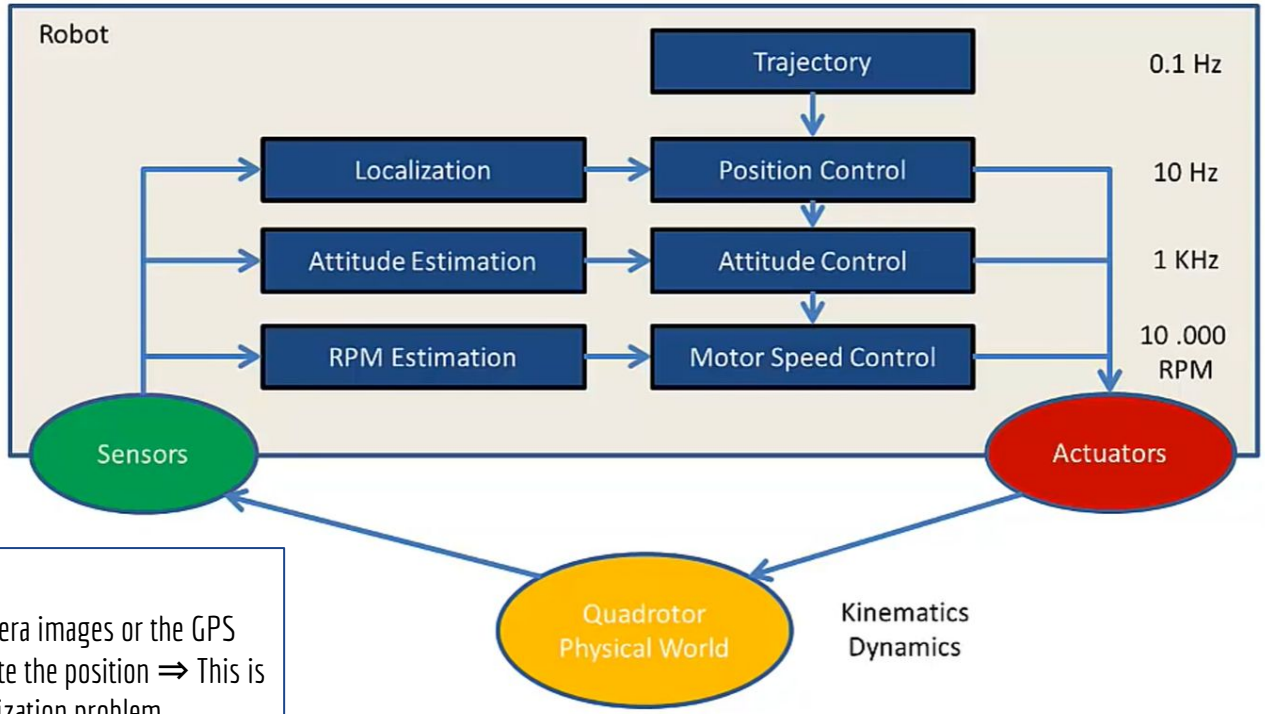
The **problem** here is if we make the gains too larges ⇒ It leads to overshooting and oscillations, and if we make them too small ⇒ It leads to long time of convergence.

The **best behavior** is to start the system with small gain, and incrementally increase them until we see a good action.

Cascaded Control

Mostly, we don't have only one controller, but a sequence of controllers.

Cascaded control ⇒ Stacking different controllers above each other.



Note

We can use camera images or the GPS signal to compute the position ⇒ This is the idea of localization problem.

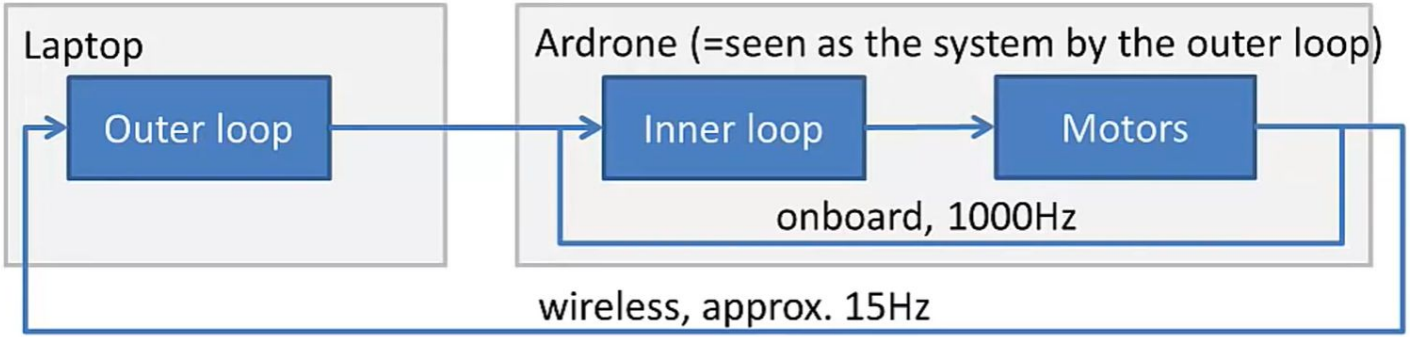
Assumptions of Cascaded Control

- Dynamics of inner loops is so fast that it is not visible from outer loops
- Dynamics of outer loops is so slow that it appears as static to the inner loops

Example: Ardrone

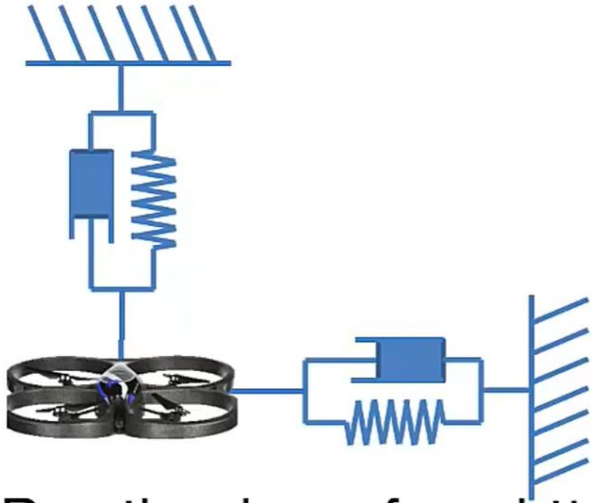
It contains an inner loop that runs in the embedded PC, which controls the attitude and the speed of the quadrotor and then the outer loop that runs on a laptop and implements the position control.

These two controls run at different speeds, the internal loop (Unknown, but may ne 1 KHZ) while the outer loop runs depending on the wireless connection at approximately at 15 to 20 KHZ.



Mechanical Equivalent

- PD Control is equivalent to adding spring-dampers between the desired values and the current position



- Run the demo from http://wiki.ros.org/tum_ardrone