

# PX4 Autopilot

Firmware Guide (Spring-22)

*Please keep editing if you find mistakes*

# Multicopter Control Architecture

# Multicopter Control Architecture

- <http://docs.px4.io/master/en/concept/architecture.html>
- [http://docs.px4.io/master/en/concept/px4\\_systems\\_architecture.html](http://docs.px4.io/master/en/concept/px4_systems_architecture.html)
- <https://www.youtube.com/watch?v=nEo4WGl4Lgc&t=118s>

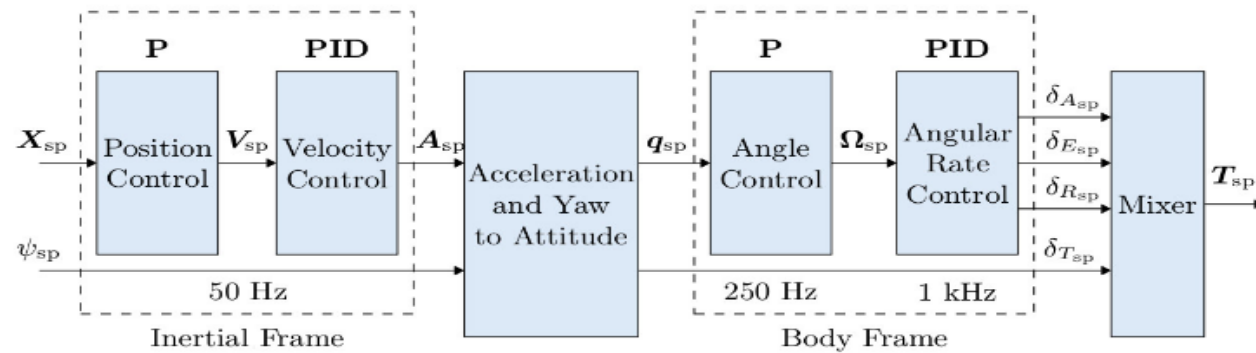
# Multicopter Control Architecture

## Controller Diagrams

This section contains diagrams for the main PX4 controllers.

The diagrams use the standard **PX4 notation** (and each have an annotated legend).

## Multicopter Control Architecture



- This is a standard cascaded control architecture.
- The controllers are a mix of P and PID controllers.
- Estimates come from **EKF2**.
- Depending on the mode, the outer (position) loop is bypassed (shown as a multiplexer after the outer loop). The position loop is only used when holding position or when the requested velocity in an axis is null.

[http://docs.px4.io/master/en/flight\\_stack/controller\\_diagrams.html](http://docs.px4.io/master/en/flight_stack/controller_diagrams.html)

# Github Repo

<https://github.com/PX4/PX4-Autopilot/blob/master/src/modules>

## ■ mc\_pos\_control

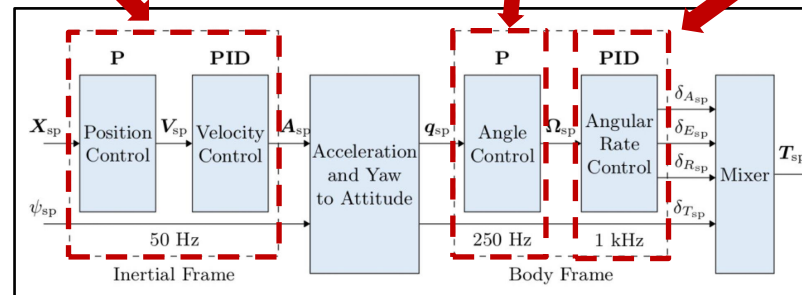
- PositionControl
  - a. PositionControl.cpp
  - b. PositionControl.hpp
- MulticopterPositionControl.cpp
- MulticopterPositionControl.hpp

## ■ mc\_att\_control

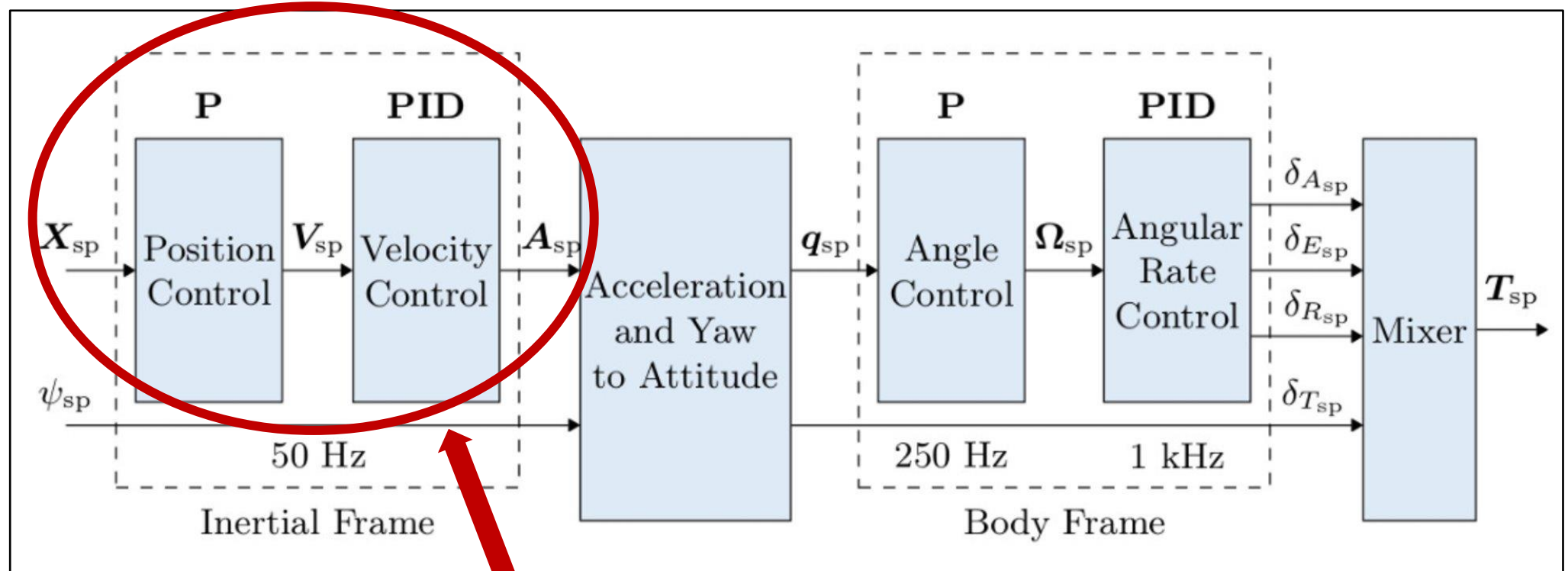
- AttitudeControl
  - a. AttitudeControl.cpp
  - b. AttitudeControl.hpp
- mc\_att\_control.cpp
- mc\_att\_control.hpp

## ■ mc\_rate\_control

- RateControl
  - a. RateControl.cpp
  - b. RateControl.hpp
- MulticopterRateControl.cpp
- MulticopterRateControl.hpp

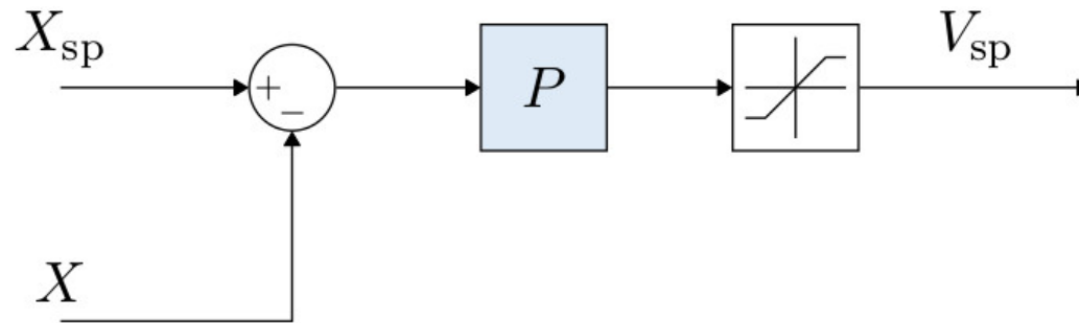


# Position & Velocity Control



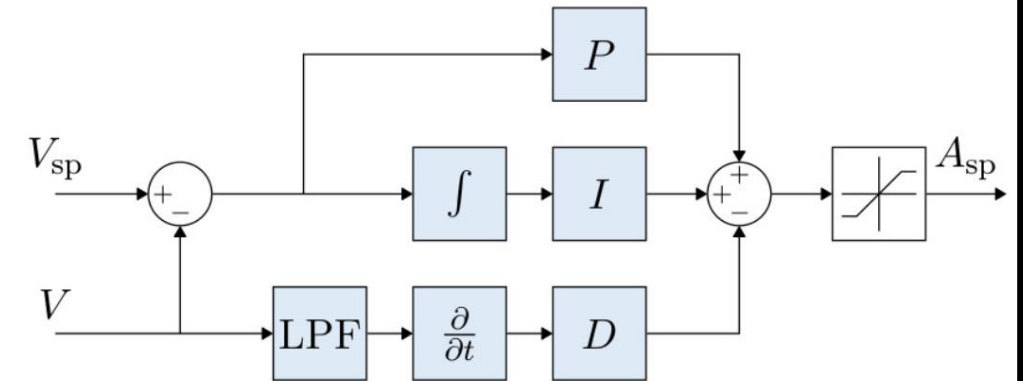
# Position & Velocity Control

Multicopter Position Controller



- Simple P controller that commands a velocity.
- The commanded velocity is saturated to keep the velocity in certain limits.

Multicopter Velocity Controller

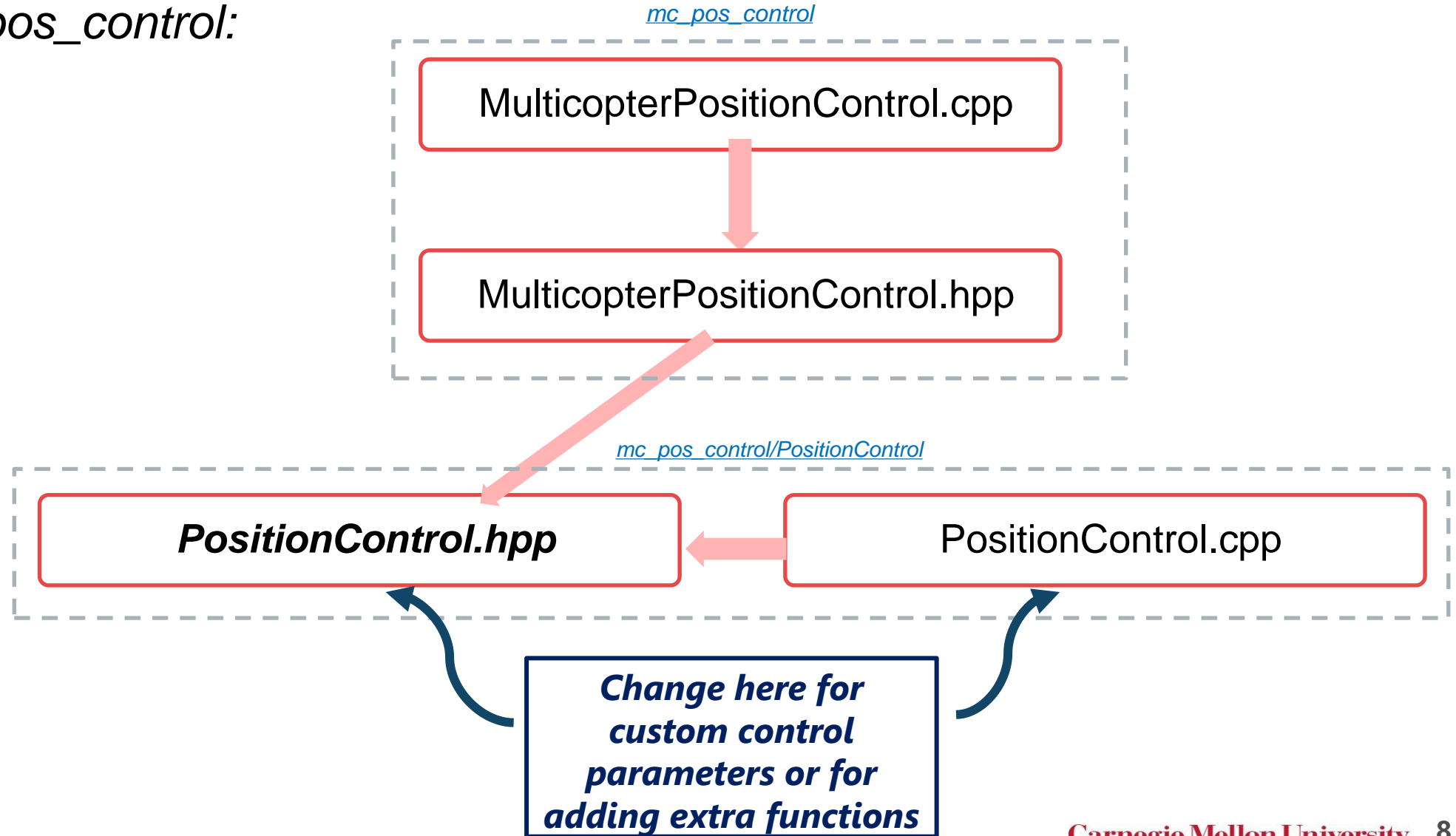


- PID controller to stabilise velocity. Commands an acceleration.
- The integrator includes an anti-reset windup (ARW) using a clamping method.
- The commanded acceleration is saturated.

[http://docs.px4.io/master/en/flight\\_stack/controller\\_diagrams.html](http://docs.px4.io/master/en/flight_stack/controller_diagrams.html)

# Codeflow

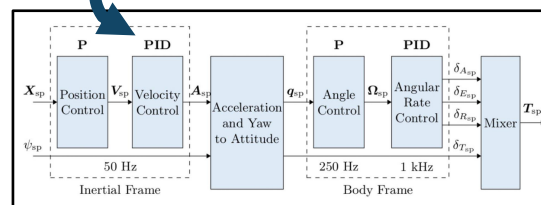
- *mc\_pos\_control*:





# Setpoint Hierarchy

- If position-setpoint && velocity-setpoint true:
  - (Velocity component of P-Controller) >>> (feed-forward component from velocity-setpoint)
- If position/velocity-setpoint && thrust-setpoint true:
  - thrust-setpoint omitted and recomputed from next cascade/step in the architecture (i.e. Velocity PID controller)



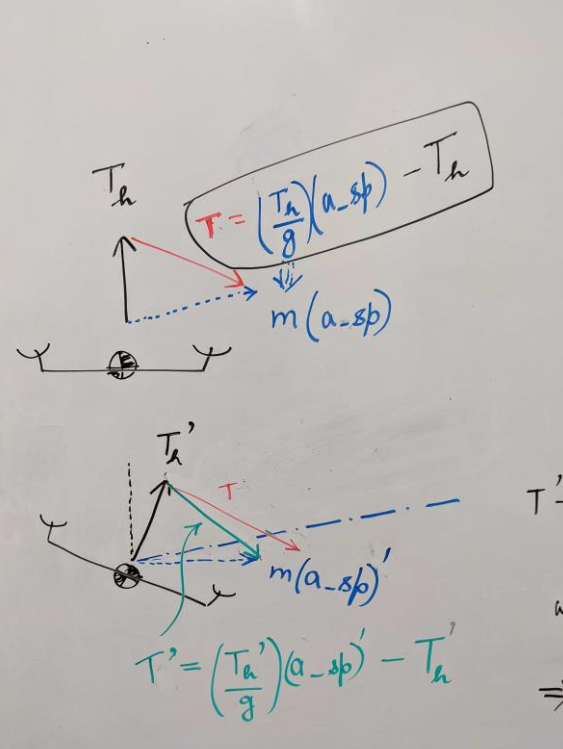
```

52  };
53
54  /**
55   *   Core Position-Control for MC.
56   *   This class contains P-controller for position and
57   *   PID-controller for velocity.
58   *   Inputs:
59   *       vehicle position/velocity/yaw
60   *       desired set-point position/velocity/thrust/yaw/yaw-speed
61   *       constraints that are stricter than global limits
62   *   Output
63   *       thrust vector and a yaw-setpoint
64   *
65   *   If there is a position and a velocity set-point present, then
66   *   the velocity set-point is used as feed-forward. If feed-forward is
67   *   active, then the velocity component of the P-controller output has
68   *   priority over the feed-forward component.
69   *
70   *   A setpoint that is NAN is considered as not set.
71   *   If there is a position/velocity- and thrust-setpoint present, then
72   *   the thrust-setpoint is omitted and recomputed from position-velocity-PID-loop.
73   */
74  class PositionControl
    
```

*mc\_pos\_control/PositionControl/PositionControl.hpp*

# Hover Thrust Update

- `void PositionControl::updateHoverThrust(const float hover_thrust_new)`
  - Line 73-85



$$T' - T = \left\{ \frac{T_h'}{g} (a_{sp}') - T_h' \right\} - \left\{ \frac{T_h}{g} (a_{sp}) - T_h \right\}$$

$$T' - T = T_h' \left( \frac{a_{sp}'}{g} - 1 \right) - T_h \left( \frac{a_{sp}}{g} - 1 \right)$$

we want  $T' - T \approx 0$

$$\Rightarrow T_h' \left( \frac{a_{sp}'}{g} - 1 \right) = T_h \left( \frac{a_{sp}}{g} - 1 \right)$$

$$\Rightarrow \frac{T_h}{T_h'} = \frac{a_{sp}' - g}{a_{sp} - g}$$

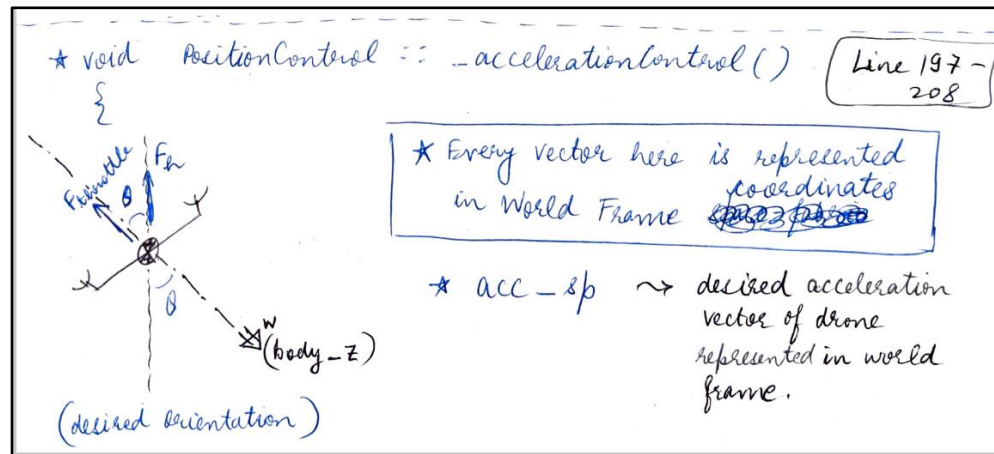
$$\Rightarrow \boxed{a_{sp}' = \frac{T_h}{T_h'} \{ a_{sp} - g \} + g}$$

New set-point

`mc_pos_control/PositionControl/PositionControl.cpp`

# Thrust Setpoint and Acceleration Setpoint (1)

- Output of Velocity-PID block
- void PositionControl::\_accelerationControl
  - Line 197-208
  - This is called earlier in void PositionControl::\_velocityControl()



★ body-Z → desired z-axis (or orientation) of drone represented in world-frame.

Line 200 ⇒  $body-Z = \begin{bmatrix} acc\_sp(0) \\ -acc\_sp(1) \\ CONSTANTS\_G \end{bmatrix}$

For  $\theta \approx small$ ;

$F_1 = F_{gravity} = F_g$  (Equilibrium along vertical <sup>world</sup> axis)

$F_2 = F_{thrust} = F_{throttle}$

~~thrust is along axis~~

$|ma_z| = |F_g| + |F_{extra}|$  } along vertical world-Z

⇒  $|F_{extra}| = -|F_g| + |ma_z|$

LINE 203 ⇒ Collective-thrust  $\equiv |F_{extra}|$

⇒  $|F_{extra}| = m|acc\_sp(2)| - |F_g|$

we need to ~~project~~ project  $|F_g|$  along drone-Z axis (represented in World Frame)

From figure;  $|F_{thrust}| \cos \theta = |F_{extra}|$

mc\_pos\_control/PositionControl/PositionControl.cpp

# Thrust Setpoint and Acceleration Setpoint (2)

- Output of Velocity-PID block
- void PositionControl::\_accelerationControl
  - Line 197-208
  - This is called earlier in void PositionControl::\_velocityControl()

③

$$\Rightarrow |F_{thrust}| = \frac{|F_{extra}|}{\cos \theta}$$

$$= \frac{|F_{extra}|}{\left\{ \frac{(\text{body-}z) \cdot [0 \ 0 \ 1]^T}{\left\| \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\|} \right\}}$$

← (Line 203)  
Line 205

Line 205  $\Rightarrow \frac{|F_{thrust}|}{\text{collective\_thrust}} = \frac{\text{collective\_thrust}}{|F_{extra}|}$  (from line 203)

Line 207  $\Rightarrow -thr\_sp = (\text{body-}z) * \text{collective\_thrust};$   
 unit vector from line 200, representing desired drone-orientation represented in the world frame!

Thrust-setpoint =  $|F_{thrust}| \begin{bmatrix} -acc\_sp(0) \\ -acc\_sp(1) \\ \text{CONSTANTS-G} \end{bmatrix}$  (P.T.O)

Overall, ④

$$-thr\_sp = \frac{\left\{ m[acc\_sp(2)] - hover\_th \right\} \begin{bmatrix} -acc\_sp(0) \\ -acc\_sp(1) \\ 9.81 \end{bmatrix}}{\left\{ \begin{bmatrix} -acc\_sp(0) \\ -acc\_sp(1) \\ 9.81 \end{bmatrix}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\}}$$

Line 252  
★ void PositionControl::getAttitudeSetpoint()  
 {  
 ControlMath::thrustToAttitude(-thr\_sp, -yaw\_sp, attitude\_setpoint);  
 attitude\_setpoint.yaw\_sp\_move\_rate = yaw\_speed\_sp;  
 }

mc\_pos\_control/PositionControl/PositionControl.cpp

# Thrust Setpoint To Rotation Matrix

- /PositionControl/ControlMath.cpp
- void thrustToAttitude()
  - Line 49
- void bodyzToAttitude()
  - Line 70 onwards
  - "R" -> Line 104 – 108:
    - *\*Mapping from body-frame coordinates to world-coordinates....according to Zac's notation ->  $X_{world} = R * X_{body}$ .*

Overall, ④

$$-thr\_sp = \frac{\{m[acc\_sp(2)] - hover\_th\} \begin{bmatrix} -acc\_sp(0) \\ -acc\_sp(1) \\ 9.81 \end{bmatrix}}{\left\{ \begin{bmatrix} -acc\_sp(0) \\ -acc\_sp(1) \\ 9.81 \end{bmatrix}^T \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right\}}$$

Line 252

```
★ void PositionControl::getAttitudeSetpoint()  
{  
    ControlMath::thrustToAttitude(-thr_sp, yaw_sp, attitude_setpoint);  
    attitude_setpoint.yaw_sp_move_rate = yaw_speed_sp;  
}
```

This is probably just a message topic

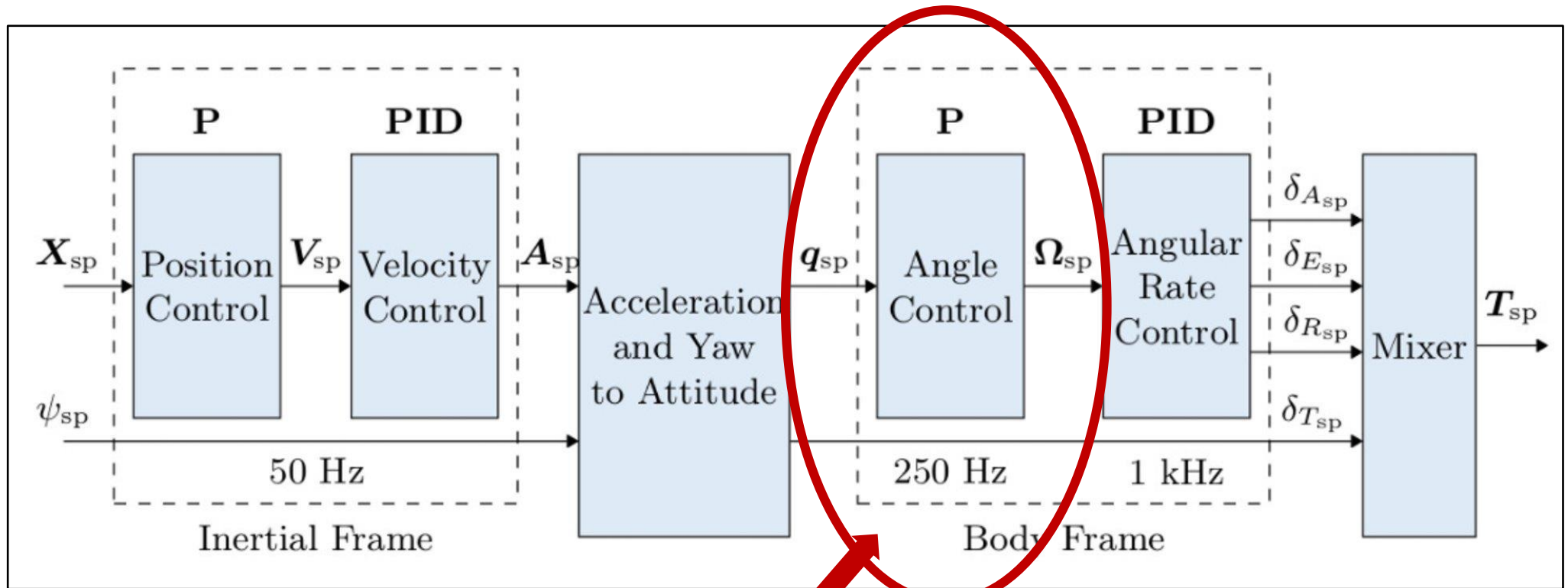
mc\_pos\_control/PositionControl/PositionControl.cpp

# How is all of this called?

- */mc\_pos\_control/MulticopterPositionControl.cpp*
- `void MulticopterPositionControl::Run()`
  - **\_control** object, throughout this file, refers to *"/PositionControl/PositionControl.cpp"*
  - Line 474 and 495 calls **\_control.update** that triggers the Position-Control algorithm!

*mc\_pos\_control/MulticopterPositionControl.cpp*

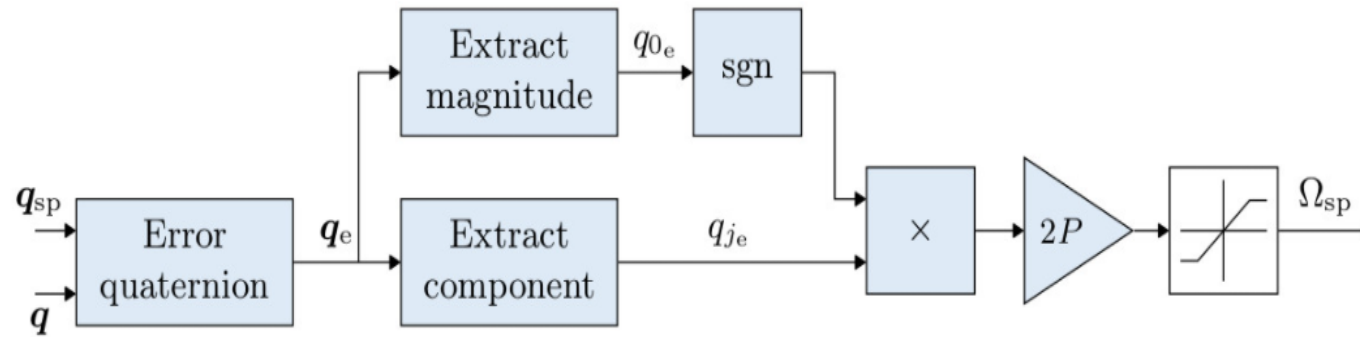
# Attitude Control





# Attitude Control

## Multicopter Attitude Controller



- The attitude controller makes use of [quaternions](#) .
- The controller is implemented from this [article](#) .
- When tuning this controller, the only parameter of concern is the P gain.
- The rate command is saturated.

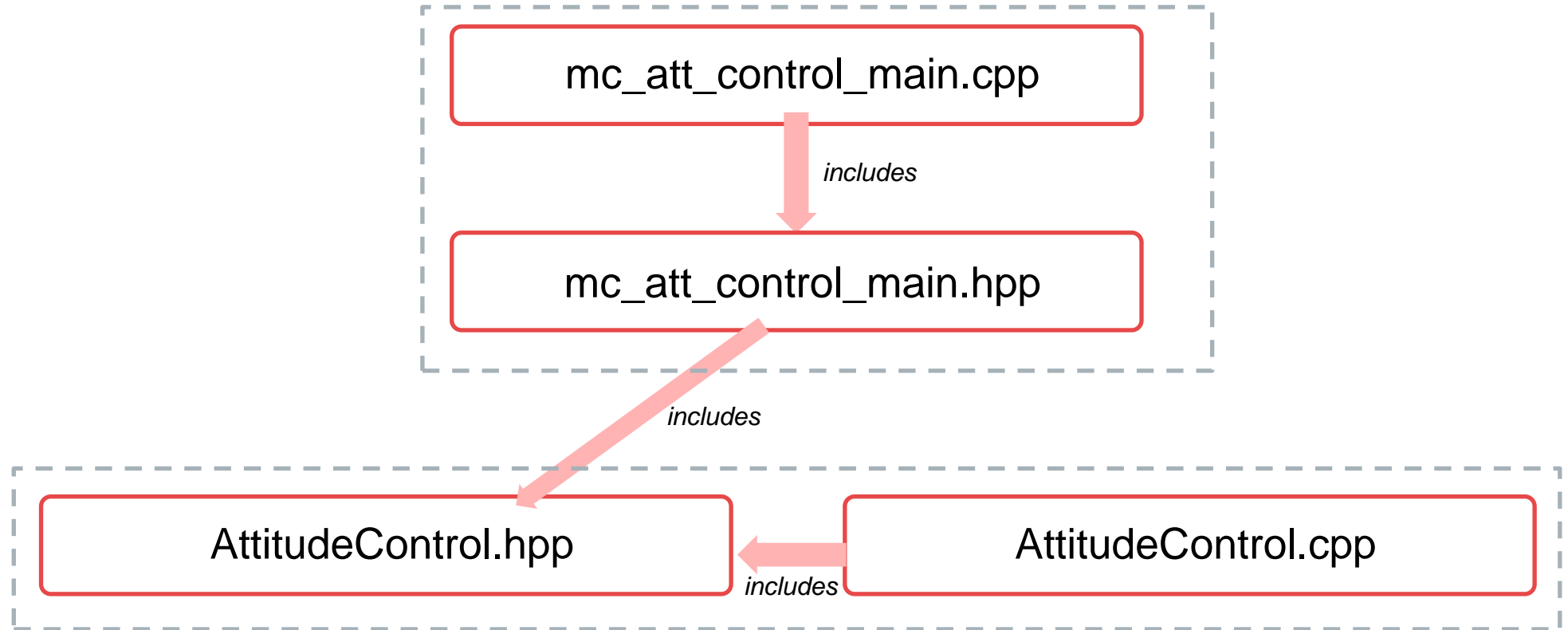
[http://docs.px4.io/master/en/flight\\_stack/controller\\_diagrams.html](http://docs.px4.io/master/en/flight_stack/controller_diagrams.html)

*\*\*Refer this link for an additional note on IMU pipeline*



# Codeflow

- *mc\_att\_control*: (quaternion based control : <https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/154099/eth-7387-01.pdf> )



**Mc\_att\_control\_main.cpp**

# generate\_attitude\_setpoint

- `/mc_att_control/mc_att_control_main.cpp`
- `void MulticopterAttitudeControl::generate_attitude_setpoint()`
  - Line 155 – 157
    - ✓ For axis angle representation, refer this: 1) [https://en.wikipedia.org/wiki/Axis%E2%80%93angle\\_representation](https://en.wikipedia.org/wiki/Axis%E2%80%93angle_representation)
    - ✓ 2) <https://github.com/Optimal-Control-16-745/lecture-notebooks-2021/blob/main/Lecture%2013/Lecture%2013.pdf>
  - Line 165, 166, 177 spits out the Roll, Pitch and Yaw setpoints -->

<ul style="list-style-type: none"><li>• <code>attitude_setpoint.roll_body</code></li><li>• <code>attitude_setpoint.pitch_body</code></li><li>• <code>attitude_setpoint.yaw_body</code></li></ul>
--
- `void MulticopterAttitudeControl::Run()`
  - **Line 304 – 308**
    - **Generates attitude setpoints if we are in Manual/Stabilized mode!**
  - Line 310 calls "`generate_attitude_setpoint()`" mentioned above
  - Line 318 calls "`_attitude_control.update(q)`". This belongs to "AttitudeControl.cpp" and is discussed in next slides

# AttitudeControl.cpp

# AttitudeControl.cpp

- */mc\_att\_control/AttitudeControl/AttitudeControl.cpp*
- `void AttitudeControl::setProportionalGain()`
- `void AttitudeControl::update()`
  - Refer <https://github.com/PX4/PX4-Autopilot/blob/master/src/lib/matrix/matrix/Quaternion.hpp>
- Half-Way Quaternion Solution: (<https://github.com/PX4/PX4-Autopilot/blob/master/src/lib/matrix/matrix/Quaternion.hpp> --> line 232)
  - <https://stackoverflow.com/questions/1171849/finding-quaternion-representing-the-rotation-from-one-vector-to-another>

*mc\_att\_control/M.cpp*

# AttitudeControl.cpp

src/modules/mc\_att\_control/AttitudeControl

AttitudeControl.cpp

→ 1) Line 60 →  $e\_z = q\_dcm()$ ;  
Find out 3<sup>rd</sup> column (z-column) of Rotation matrix representing current attitude of drone from quaternion.

→ 2) Line 61 →  $e\_z\_d = q\_dcm\_z()$ ;  
↳ attitude setpoint!  
Find out 3<sup>rd</sup> column of Rotation matrix that represents desired/setpoint attitude of the drone from the setpoint-quaternion.

→ 3) Line 62 →  $Quatf\ q\_d\_red(e\_z, e\_z\_d)$   
↳ Refer to src/lib/matrix/matrix/Quaternion.hpp.  
Find out the quaternion that has the information of rotation from "e\_z" vector to "e\_z\_d" vector. This represents rotation from current attitude (for e.g. @ time step 'k') to desired attitude (@ time-step 'k+n')

→ 4) Line 72 →  $q\_d\_red = (q\_d\_red) \otimes (q)$ ;  
Final (desired) attitude quaternion with respect to world frame.  
change in attitude (or rotation) from current to desired attitude.  
current attitude quaternion with respect to world frame.

NOTE: → 1), 2), 3), 4) ignore yaw & prioritize Roll+Pitch!

→ 5) Line 76 →  $q\_mix = q\_d\_red.inversed() \otimes q\_d$ ;  
This will probably have errors(?)  
Then apply reverse rotation using q\_d\_red that we found out in line 72. [Ideally, should probably give us the exact world frame orientation]  
First go to the desired/set-point thrust-vector (or attitude representation)

→ 6) Line 79 & 80 → specifically eliminating numerical anomalies in yaw? ⇒ For e.g. any rotation about yaw axis  
↳ 
$$\begin{bmatrix} \cos(\frac{\psi}{2}) & 0 & 0 \\ 0 & \cos(\frac{\psi}{2}) & 0 \\ 0 & 0 & \sin(\frac{\psi}{2}) \end{bmatrix}$$

→ 7) Line 81 →  $q\_d = q\_d\_red \otimes Quatf(, 0, 0, )$ ;  
Final desired attitude  
Then, apply rotation that takes you to the current 'q' and then to desired attitude.  
First apply some rotation in yaw. (Probably error compensation???)

NOTE: → In 5) & 6) & 7) we almost did the same thing except we considered yaw somehow...

[https://github.com/PX4/PX4-Autopilot/blob/master/src/modules/mc\\_att\\_control/AttitudeControl/AttitudeControl.cpp](https://github.com/PX4/PX4-Autopilot/blob/master/src/modules/mc_att_control/AttitudeControl/AttitudeControl.cpp)

# AttitudeControl.cpp

8) Line 84  $\rightarrow q_e = q \cdot \text{inversed}() \otimes q_d$  ;

*\* error-quaternion \**

*This inverse rotation*

*First rotate from world frame to the desired attitude (or thrust vector)  $q_d$ . (NOTE  $\rightarrow$  This  $q_d$  was calculated from line 81)*

*They seem to have applied  $q_e$  first, then  $q$  (?)*

$$q \otimes q_e = q_d$$

$$\Rightarrow q^{-1} \otimes q \otimes q_e = q^{-1} \otimes q_d$$

$$\Rightarrow q_e = q^{-1} \otimes q_d$$

9) Line 88  $\rightarrow eq = (-----)$

Extract vector (imaginary) part of the error quaternion

$\rightarrow$  which represents the axis-angle representation of the change in orientation.

$\rightarrow$  This axis-angle is used as "error" term to be multiplied by Proportional ~~From~~ Gain!

10) Line 93-101 :  $_{-}yaw\ speed\_setpoint$  comes from the commander and this is represented in the world frame (i.e. about world-Z)

The topic  $_{-}rates\_sp$  publishes rates expressed in Body frame.

$\star$  (Read lines 93-99)  $\star$

$$\rightarrow rate\_setpoint = [K_{p1} \ K_{p2} \ K_{p3}] \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix} + \{q \cdot \text{inversed}() \cdot \text{dom\_z}\}$$

*Proportional Controller applied to error-term (axis angle extracted from error-quaternion)*

*extract z-column from (Rotation Matrix)<sup>T</sup> i.e. extract the world-Z axis expressed as seen from the Drone-Body Frame.*

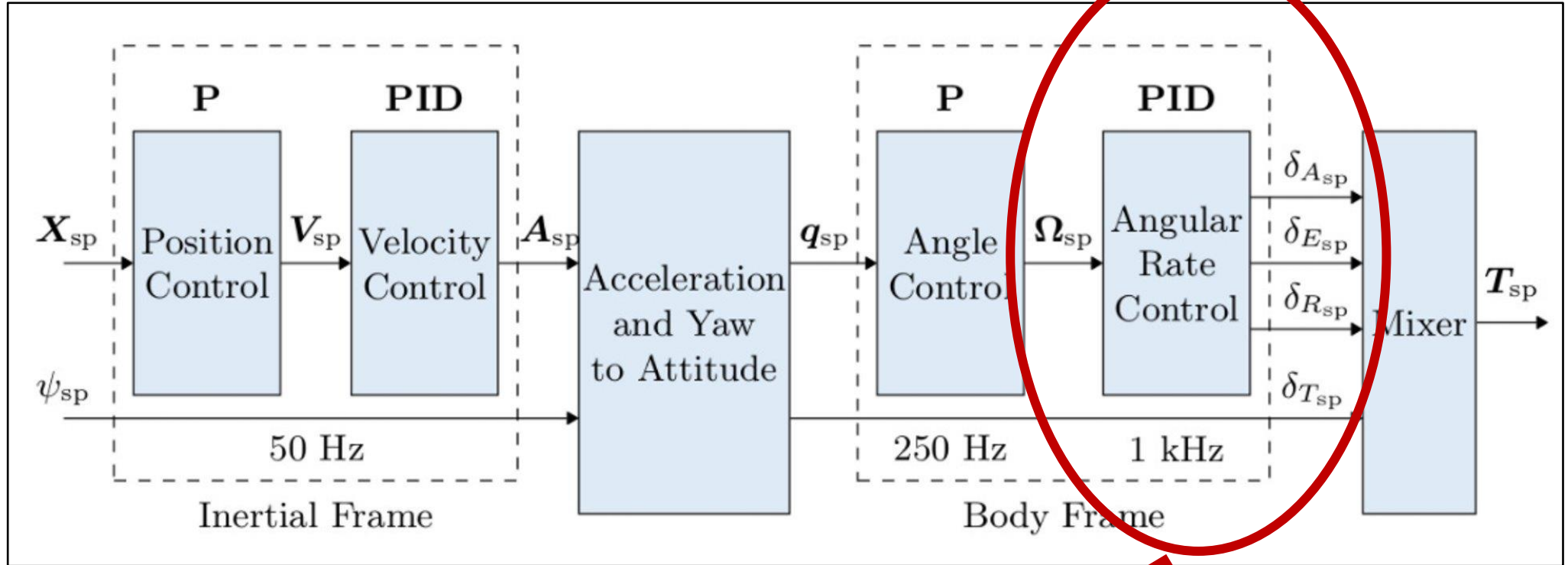
*\* yaw speed setpoint \**

# Quaternion Starter-Pack

- <https://www.youtube.com/watch?v=zjMulxRvygQ>
- <https://github.com/Optimal-Control-16-745/lecture-notebooks-2021/blob/main/Lecture%2013/Lecture%2013.pdf>
- <https://github.com/Optimal-Control-16-745/lecture-notebooks-2021/blob/main/Lecture%2014/Lecture%2014.pdf>
- <https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/154099/eth-7387-01.pdf>
- <https://ieeexplore.ieee.org/document/9326337>

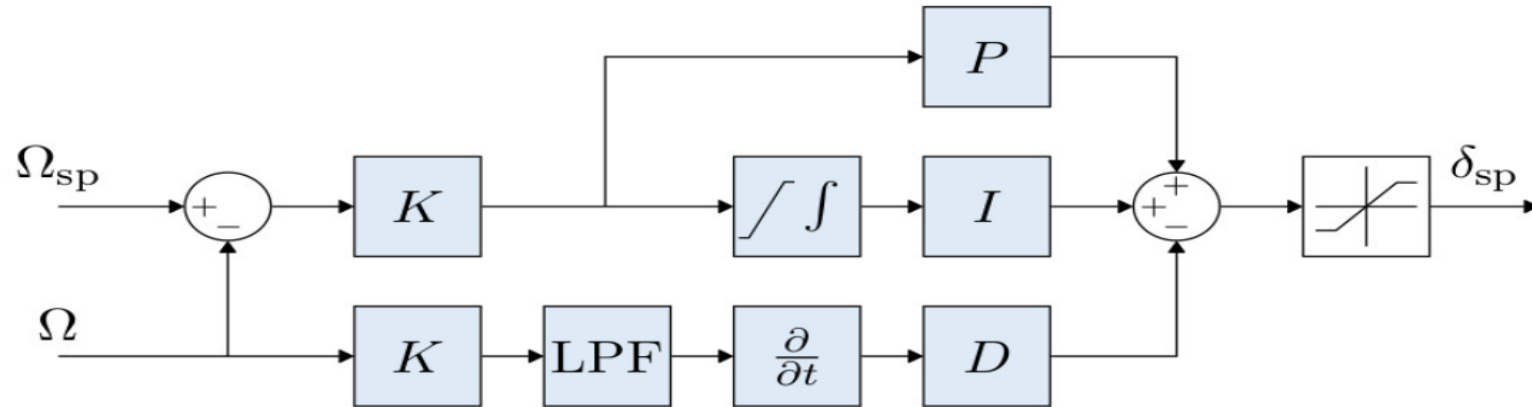


# Angular Rate Control



# Angular Rate Control

Multicopter Angular Rate Controller



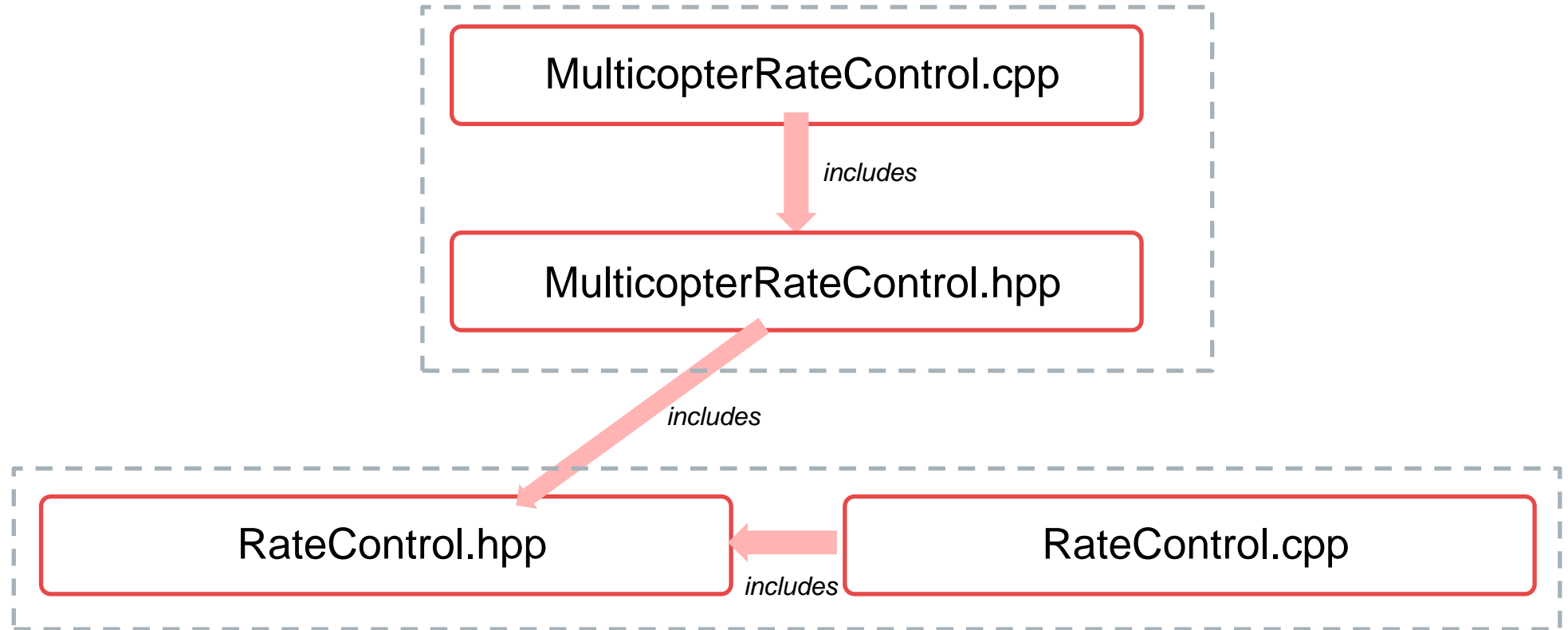
- K-PID controller. See [Rate Controller](#) for more information.
- The integral authority is limited to prevent wind up.
- The outputs are limited (in the mixer), usually at -1 and 1.
- A Low Pass Filter (LPF) is used on the derivative path to reduce noise (the gyro driver provides a filtered derivative to the controller).

[http://docs.px4.io/master/en/flight\\_stack/controller\\_diagrams.html](http://docs.px4.io/master/en/flight_stack/controller_diagrams.html)

*\*\*Refer this link for an additional note on IMU pipeline*

# Codeflow

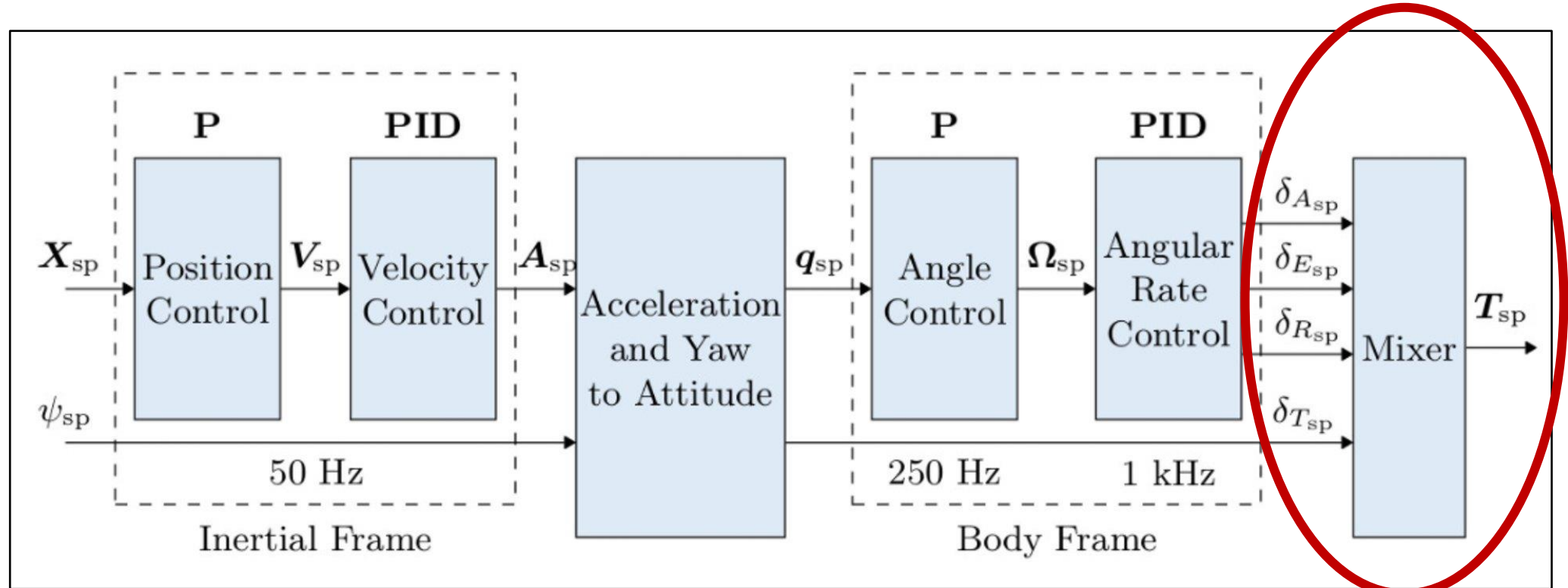
- *mc\_rate\_control*: (quaternion based control : <https://www.research-collection.ethz.ch/bitstream/handle/20.500.11850/154099/eth-7387-01.pdf> )



# MulticopterRateControl.cpp

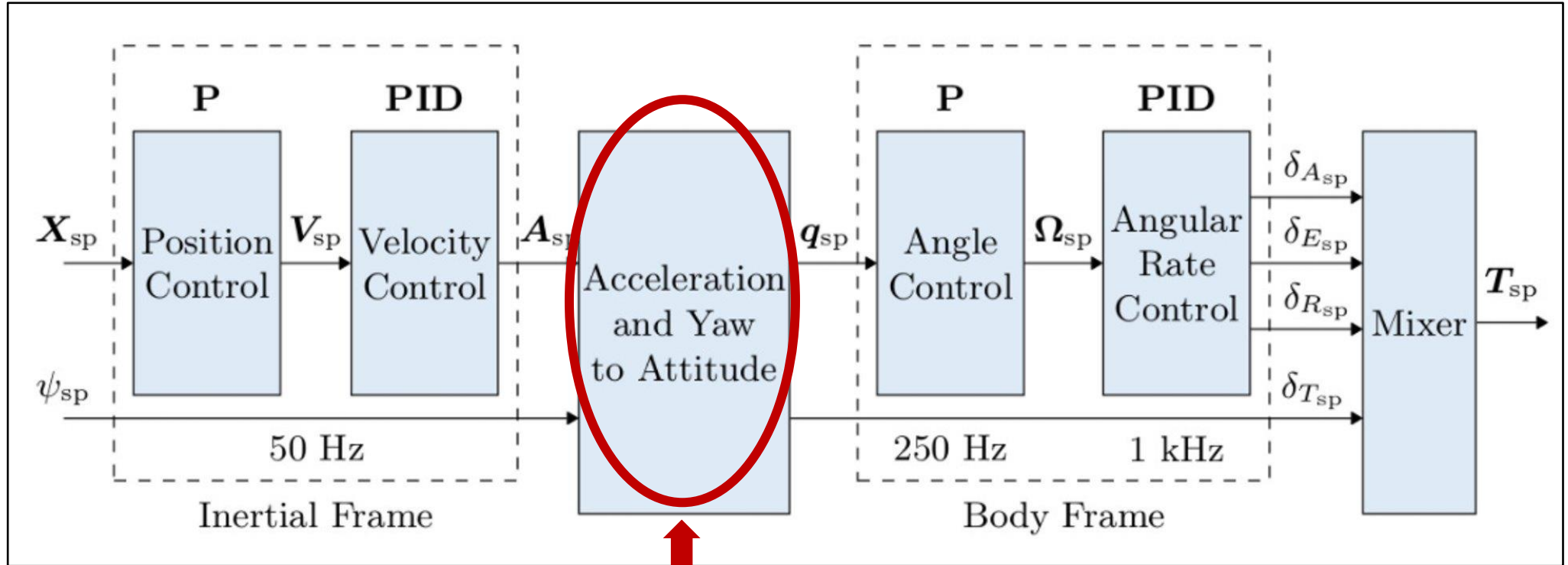
# RateControl.cpp

# Mixing and Actuators



- <http://docs.px4.io/master/en/concept/mixing.html>
- [https://github.com/PX4/PX4-Autopilot/blob/master/src/drivers/pwm\\_out/PWMOut.cpp](https://github.com/PX4/PX4-Autopilot/blob/master/src/drivers/pwm_out/PWMOut.cpp)

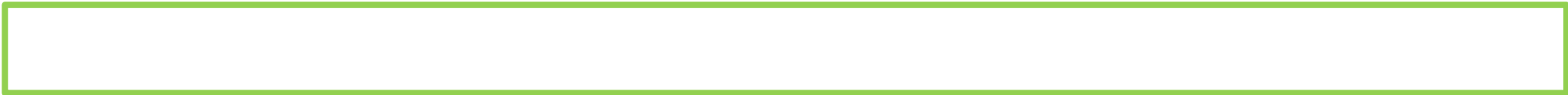
# Attitude Setpoint



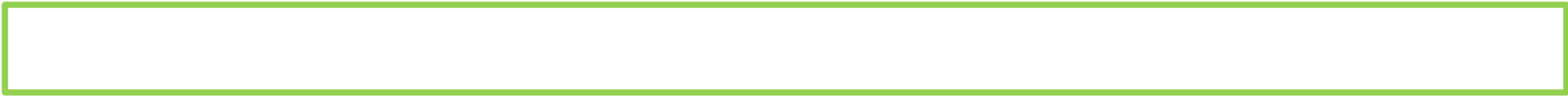
# UUV Control Architecture



# Coming soon...



***\*\*The following slides are for  
shapes/arrows etc.\*\****

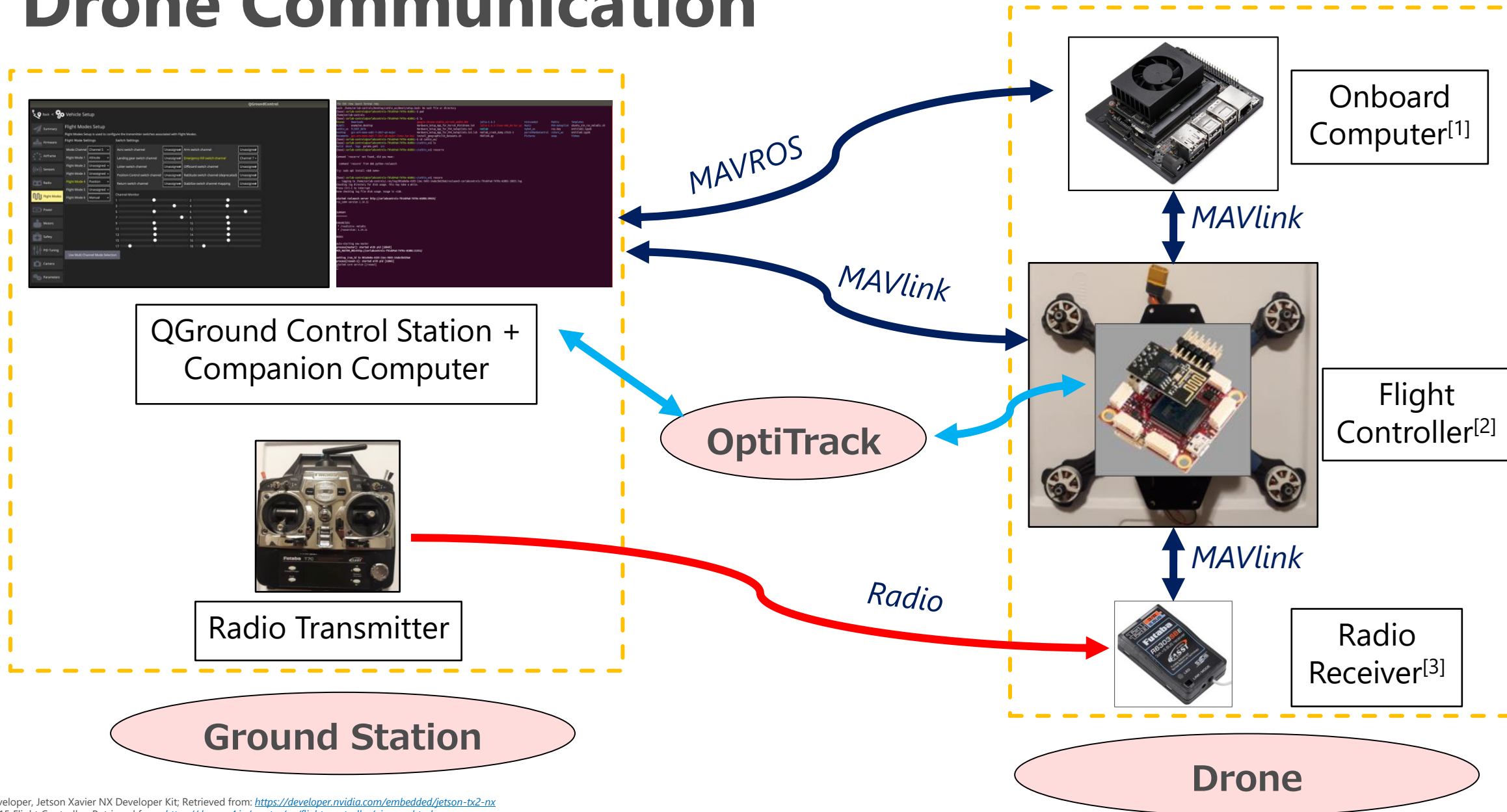


# T

- 2 DIY TBS drones ready
- Relevance?
  - Experience/knowledge transfer to BlueROV platform
- Custom Firmware
  - Need to understand PX4 Codebase first



# Drone Communication



[1] Nvidia Developer, Jetson Xavier NX Developer Kit; Retrieved from: <https://developer.nvidia.com/embedded/jetson-tx2-nx>

[2] Pixracer-R15 Flight Controller; Retrieved from: [https://docs.px4.io/master/en/flight\\_controller/pixracer.html](https://docs.px4.io/master/en/flight_controller/pixracer.html)

[3] Futaba Radio Receiver; Retrieved from: [https://www.bhphotovideo.com/c/product/1507099-REG/futaba\\_01102196\\_1\\_r6303sbe\\_fasst\\_s\\_bus\\_2\\_4.html?ap=y&ap=y&smp=y&smp=y&lsft=BI%3A514&qclid=CjwKCAjwoZWHBhBgEiwAiMN66b4V0cBzoYgmVTPghbsPP8sguX42iL51vwlocrRU7q7cbRuSYWUjRoCSE0QAvD\\_BwE](https://www.bhphotovideo.com/c/product/1507099-REG/futaba_01102196_1_r6303sbe_fasst_s_bus_2_4.html?ap=y&ap=y&smp=y&smp=y&lsft=BI%3A514&qclid=CjwKCAjwoZWHBhBgEiwAiMN66b4V0cBzoYgmVTPghbsPP8sguX42iL51vwlocrRU7q7cbRuSYWUjRoCSE0QAvD_BwE)

# Ongoing and Future Work

## Hardware

- • Mount Jetson Nano onboard
- • Mount Wifi-modules/Radio telemetry modules if necessary
- • Reassess hardware design because of increased payload
- Brainstorm Tether integration

## Software

- • PX4 Codebase
- • ROS offboard control
- • OptiTrack integration
- Custom Control Architecture flashing
- Reinforcement Learning Research



# M

- Cheap Mattress → 35-50\$  
each

- Gymnastic mats → 300-400\$  
each

