A Report for Bounding Function Taxonomy

A. Engine Management

The heart of UAVs are the engines. For motor-based engines, the force it provides is correlated with its number of *Revolutions Per Minute* (RPM). UAV systems typically quantify the force through the *throttle* and the *thrust*. Strictly speaking, the throttle is an engine component that manipulate the thrust output (i.e., the force), but the two are often used interchangeably in the context of UAVs. In Paparazzi, throttle/thrust is represented by a value ranging between [0, 1], with 1 aligned with our intuition of “full force.” We identified 21 instances of BFs applied for engine management, over the physical variables of thrust/throttle and RPM. They are divided into the following use scenarios.

1) Safe Landing: (a) Use Context: A ”smooth” landing of the UAV coincides with the exponential reduction of both height and vertical velocity. According to optic flow theory [1], this can be achieved through maintaining a constant *flow divergence*, which can be effectively managed by thrust control. (b) Datatype: thrust/throttle. (c) The Need for BFs: as landing generally requires a reduction on the thrust, the upper bound is easily understood. The lower bound, however, is equally important: an excessively low thrust may cause stall, a catastrophic event that UAVs do not have sufficient lift to counter gravity. (d) Example: In this code snippet [1], the thrust value output by the PID controller for optic flow is bounded within the range of

[0.25 * of_landing_ctrl.nominal_thrust * MAX_PPRZ, MAX_PPRZ]

where of_landing_ctrl.nominal_thrust is the nominal thrust around which the PID control operates, and MAX_PPRZ is a constant that can be customized by Paparazzi users. (e) Occurrence: 7 instances.

2) Safe Motor Mixing: (a) Use Context: For rotorcraft with at least 2 rotors, the throttle of each motor is managed by its controller, and “mixed” together according to the layout among the motors [2] to compute the overall throttle of the UAV. (b) Datatype: thrust/throttle. (c) The Need for BFs: Unlike single motors where the change of controller output can be managed by the PID/INDI-like control algorithms themselves, the output of motor mixing may experience rapid change. Drastic change of UAV thrust/throttle may cause instability for the maneuver of UAVs. (d) Example: In this code snippet [2], the change in throttle by consecutive motor mixing outputs is represented by a variable named saturation_offset, which is in turn bound to a range. (e) Occurrence: 8 instances.

3) Safe Motor Speed Change: (a) Use Context: Motor speed is correlated with the thrust/throttle it may produce. (b) Datatype: RPM. (c) The Need for BFs: A drastic change in motor speed may not only cause a drastic change in thrust (which will cause instability for UAVs as we described earlier), but also may cause damage to the motor itself. In manually controlled UAVs, users may adjust the throttle curve settings, a discrete representation of different levels of throttle. The change in motor speed must be bounded to avoid excessive user-inputed change. (d) Example: In this code snippet [3], the change in RPM is represented by a variable named rpm_diff, which is in turn bound to the range of

[-THROTTLE_CURVE_RPM_INC_LIMIT / 512,
THROTTLE_CURVE_RPM_INC_LIMIT / 512],

where THROTTLE_CURVE_RPM_INC_LIMIT is a customizable limit for RPM increment. (e) Occurrence: 5 instances.

4) Collision Avoidance: (a) Use Context: Obstacle and collision avoidance is an essential safety-critical concern of UAVs. In Paparazzi, UAV obstacle avoidance is based on potential field [3]. A throttle control using potential field is utilized to compute the desired throttle/thrust to avert aircraft collision. (b) Datatype: throttle/thrust. (c) The Need for BFs: A high throttle level would lead to a overly large speed of the UAV. In this case, a collision may occur because of the inertia of the UAV even though potential field is used for avoidance. (d) Example: In Listing 1, the desired throttle level cruise computed using potential field is bounded. Note that even though the variables here may indicate cruise is related to cruise speed, the semantics beneath the variable is indeed a throttle value. For example, constants V_CTL_AUTO_THROTTLE_MIN_CRUISE_THROTTLE and V_CTL_AUTO_THROTTLE_MAX_CRUISE_THROTTLE in the default implementation of Paparazzi are set at 0.2 and 1.0, respectively. (e) Occurrence: 1 instance.

```
int potential_task(void)
{
  ...

  // speed loop
  float cruise =
     V_CTL_AUTO_THROTTLE_NOMINAL_CRUISE_THROTTLE;
  cruise += -force_speed_gain * (potential_force.north * ch
     + potential_force.east * sh);
  Bound(cruise, V_CTL_AUTO_THROTTLE_MIN_CRUISE_THROTTLE,
     V_CTL_AUTO_THROTTLE_MAX_CRUISE_THROTTLE);
  potential_force.speed = cruise;
```

[1]Function PID_divergence_control at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/modules/ctrl/optical_flow_landing.c

[2]Function motor_mixing_run at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/subsystems/actuators/motor_mixing.c

[3]Function throttle_curve_run at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/modules/helicopter/throttle_curve.c

```
  v_ctl_auto_throttle_cruise_throttle = cruise;

  ...
}
```

Listing 1: An Example on Throttle/Thrust Bounding for Collision Avoidance[4]

B. *Speed and Acceleration Management*

The force generated by the engine directly impact the *acceleration* of the UAVs, which in turn impacts their *speed*. We identified 17 instances of BFs applied for speed and acceleration management. They are divided into 4 use scenarios.

1) Safe Acceleration for Navigation: (a) Use Context: A positive acceleration is needed for speeding up the UAV, whereas a negative one manages to slow it down. Acceleration is essential for the take-off and landing of UAVs. (b) Datatype: acceleration. (c) The Need for BFs: an extremely high value of the acceleration may negatively impact the stability of UAVs. (d) Example: In this code snippet [5], horizontal acceleration in the two-dimensional reference model, `gh_ref.accel`, is bounded on the X and Y directions respectively. The bounds are set by another variable, `gh_ref.max_accel`. This is an example where the bound is not a constant but another dynamic program value, i.e., a *dynamic bound*. (e) Occurrence: 4 instances.

2) Safe Acceleration Request as Engine Input: (a) Use Context: As acceleration must be realized by engine throttle/thrust changes, different navigation program modules may request a desirable acceleration to the control algorithm of the engine itself. (b) Datatype: acceleration. (c) The Need for BFs: an excessive input may require the engine to produce a large throttle/thrust, causing a flame-out in the engine. (d) Example: In this code snippet [6], the target vertical acceleration is represented by variable `v_ctl_desired_acceleration`, and it is bounded within [`-v_ctl_max_acceleration`, `v_ctl_max_acceleration`], where `v_ctl_max_acceleration` is the dynamic upper bound for a UAV's vertical acceleration. (e) Occurrence: 8 instances.

3) Safe Wind Speed: (a) Use Context: While operating in the physical environments, UAVs need to consider the wind speed while calculating its ground speed, an important parameter that impacts its navigation, take-off, and landing. (b) Datatype: speed. (c) The Need for BFs: Wind can be "gusty" in nature, so that the measurement received from sensors may be unreliable. If the measured wind speed is excessive, it may severely impact how navigation trajectory is computed, and worse, jeopardize the safety of take-off and landing. (d) Example: Listing 2 shows a code snippet for UAV landing. Here the ground speed is calculated with wind speed in consideration. The wind speed `wind_on_final` is bounded within the range [`-MAX_WIND_ON_FINAL`, `MAX_WIND_ON_FINAL`] at line 6. If bounding was not performed, the ground speed can be excessive, impacting the decision making process during landing. (e) Occurrence: 2 instances.

```
1  static inline bool gls_compute_TOD(uint8_t _af, uint8_t _sd
     , uint8_t _tod, uint8_t _td)
2  {
3    ...
4
5    float wind_on_final = wind_norm * (((td_af_x * wind->y) /
       (td_af * wind_norm)) + ((td_af_y * wind->x) / (td_af *
       wind_norm)));
6    Bound(wind_on_final, -MAX_WIND_ON_FINAL,
         MAX_WIND_ON_FINAL);
7    gs_on_final = target_speed - wind_on_final;
8
9    ...
10 }
```

Listing 2: An Example on Wind Speed Bounding [7]

4) Safe Remote User Speed Input: (a) Use Context: For public safety reasons, UAVs with an autopilot board may often be required to operate with operator-guided remote control through governmental regulations. (b) Datatype: speed. (c) The Need for BFs: The operator may manually input a speed change that may impact the stability of a UAV. (d) Example: In this code snippet [8], a variable `guidance_v_rc_zd_sp` represents the vertical speed change inputted by the operator from remote control. It is bounded using a *deadband*, i.e., the safety threshold, as

[`-GUIDANCE_V_CLIMB_RC_DEADBAND`, `GUIDANCE_V_CLIMB_RC_DEADBAND`],

(e) Occurrence: 3 instances

C. *Pose Management*

Maintaining and maneuvering the pose of UAVs is essential for supporting key behaviors of UAVs, from take-off, to hover, to turn, to landing. The physical variables that represent the pose are *pitch*, *roll*, and *yaw*. When the change of the pose matters, pose management also considers the *rate* or *interval* of change for the three physical variables. We identified 52 instances of BFs applied for pose management, divided into 4 use scenarios.

1) Safe Pose Maintenance: (a) Use Context: Maintaining a pose plays a central role in UAV flight tasks such as follows. First, the take-off and landing requires a positive/negative value on pitch. Second, hover control, i.e., maintaining the position of a UAV in the 3-D space, is achieved through continuous correction on the pose through controllers. Third, obstacle avoidance also requires a change in pose, so that the

[4]Function `potential_task` at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/modules/multi/potential.c

[5]Function `gh_saturate_ref_accel` at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/firmwares/rotorcraft/guidance/guidance_h_ref.c

[6]Function `v_ctl_climb_loop` at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/firmwares/fixedwing/guidance/energy_ctrl.c

[7]Function `gls_compute_TOD` at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/modules/nav/nav_gls.c

[8]Function `guidance_v_read_rc` at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/firmwares/rotorcraft/guidance/guidance_v.c

trajectory may change as a result. (b) Datatype: pitch/roll/yaw. (c) The Need for BFs: Not all poses are safe for UAVs from an aerodynamic perspective. For instance, excessive pitch may cause stall. As another example, excessive roll may cause the UAV to flip. (d) Example: In this code snippet [9], the roll and pitch outputs of the obstacle avoidance algorithm are bounded by [-CMD_OF_SAT, CMD_OF_SAT], where CMD_OF_SAT is the maximum angle. (e) Occurrence: 12 instances.

2) Safe Turn Coordination: (a) Use Context: a concrete instance of safe pose maintenance, but with a unique constraint, is UAV turn coordination. When a UAV makes a turn, its roll value must be adjusted so that the UAV "banks" to an angle. (b) Datatype: roll. (c) The Need for BFs: a unique problem with turning is *side-slipping*: if a roll angle is excessive, the UAV may move sideways rather than a smooth turn. (d) Example: In this code snippet [10], the target roll angle is bounded within the range of [-h_ctl_roll_max_setpoint, h_ctl_roll_max_setpoint]. (e) Occurrence: 1 instance.

3) Safe Pose Change Rate: (a) Use Context: In UAV pose management, an attitude reference model is used to estimate the pose of the UAV and calculate the desired angular rate of change. In Paparazzi, INDI is commonly used for this purpose. (b) Datatype: pitch/roll/yaw change rate. (c) The Need for BFs: a drastic change in pose may impact the stability of the UAV. (d) Example: In the code snippet in Listing 3, the desired angular rate on yaw direction rate_ref_r is bounded into the range

[-indi.attitude_max_yaw_rate,
indi.attitude_max_yaw_rate],

where indi.attitude_max_yaw_rate is the maximum yaw rate in pose control. (e) Occurrence: 37 instances.

```c
static inline void stabilization_indi_calc_cmd(int32_t
    indi_commands[], struct Int32Quat *att_err, bool
    rate_control)
{
  ...

  //This separates the P and D controller and lets you
    impose a maximum yaw rate.
  float rate_ref_r = indi.reference_acceleration.err_r *
    QUAT1_FLOAT_OF_BFP(att_err->qz) / indi.
    reference_acceleration.rate_r;
  BoundAbs(rate_ref_r, indi.attitude_max_yaw_rate);
  indi.angular_accel_ref.r = indi.reference_acceleration.
    rate_r * (rate_ref_r - rates_for_feedback.r);

  ...
}
```

Listing 3: An Example on Safe Pose Change[11]

4) Safe Pose Change Time Interval: (a) Use Context: A variant of the use scenario described in § -C3 is that pose change may be quantified by the time interval between two adjustments. (b) Datatype: pitch/roll/yaw change time interval. (c) The Need for BFs: same as the use scenario described in § -C3. (d) Example: In Listing 4, for instance, dt represents the time interval since last call of the current function. Before used to compute the yaw angle sp->psi at line 12, it is bounded within 0.5 seconds to ensure a moderate increment of the target angle. (e) Occurrence: 2 instances.

```c
1  void stabilization_attitude_read_rc_setpoint_eulers(struct
     Int32Eulers *sp, bool in_flight, bool in_carefree, bool
      coordinated_turn)
2  {
3    ...
4
5    /* calculate dt for yaw integration */
6    float dt = get_sys_time_float() - last_ts;
7    /* make sure nothing drastically weird happens, bound dt
       to 0.5sec */
8    Bound(dt, 0, 0.5);
9
10   /* do not advance yaw setpoint if within a small deadband
       around stick center or if throttle is zero */
11   if (YAW_DEADBAND_EXCEEDED() && !THROTTLE_STICK_DOWN()) {
12     sp->psi += get_rc_yaw() * dt;
13     INT32_ANGLE_NORMALIZE(sp->psi);
14   }
15
16   ...
17 }
```

Listing 4: An Example on Safe Pose Change Time Interval [12]

D. Trajectory Management

To follow a trajectory, the UAV needs to follow waypoints, including turning occasionally (in the horizontal direction) and changing altitude (in the vertical direction). The physical variables related to trajectory management are *distance* and *heading* (change). We identified 11 instances of BFs applied for trajectory management, which we divide into 4 use scenarios.

1) Safe Homing: (a) Use Context: After performing the flight task, the UAV should go back to the ground station. (b) Datatype: distance (X and Y axis). (c) The Need for BFs: to avoid catastrophic consequences due to battery drain, a UAV (generally) should not fly too far way from the ground station. (d) Example: In this code snippet [13] , the distance between the UAV waypoint and the home waypoint is computed, and bounded by variable max_dist_from_home, the maximum distance between them. (e) Occurrence: 3 instances.

2) Safe Altitude Change: (a) Use Context: UAV systems fly in a 3-D space; altitude change is a basic task. (b) Datatype: distance (Z axis). (c) The Need for BFs: a drastic change in altitude may affect the stability of the UAV. (d) Example: In this code snippet [14] , the altitude change between two iterations

[9]Function OA_update at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/modules/obstacle_avoidance/guidance_OA.c

[10]Function gvf_control_2D at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/modules/guidance/gvf/gvf.c

[11]Function stabilization_indi_calc_cmd at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/firmwares/rotorcraft/stabilization/stabilization_indi_simple.c

[12]Function stabilization_attitude_read_rc_setpoint_eulers at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/firmwares/rotorcraft/stabilization/stabilization_attitude_rc_setpoint.c

[13]Function nav_move_waypoint at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/subsystems/navigation/common_nav.c

[14]Function gv_update_ref_from_zd_sp at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/firmwares/rotorcraft/guidance/guidance_v_ref.c

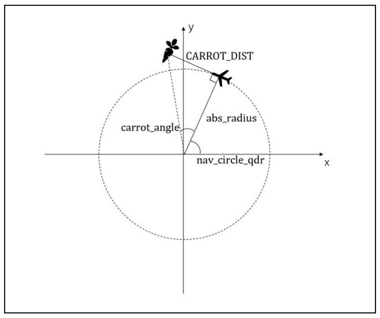

Fig. 1: Carrot-Based Guidance for Heading Change

of the control loop is bounded by `GV_MAX_Z_DIFF`, the maximum distance between the previous waypoint and the current waypoint on the Z axis. (e) Occurrence: 1 instance.

3) Safe Heading Change in Guidance: (a) Use Context: Paparazzi follows the widely used *carrot-based* approach [4] for trajectory management: a virtual, continuously updated waypoint not far from the current position of the UAV to guide the next "leg" of movement of the UAV, similar to using a carrot to attract a mule to move forward. As shown in Figure 1, the carrot-based guidance implemented by Paparazzi for circle navigation assumes a constant distance between the current position of the UAV and the carrot, as `CARROT_DIST`. By adjusting the `carrot_angle`, the UAV may change its heading. (b) Datatype: heading (change). (c) The Need for BFs: a drastic change in heading may affect the stability of the UAV, and affects the correctness of the circle trajectory. (d) Example: In Listing 5 which concerns circle navigation, the `carrot_angle` is bounded to the range of $[\frac{\pi}{16}, \frac{\pi}{4}]$. The rest of the variables are illustrated in Figure 1. (e) Occurrence: 3 instances.

```
void nav_circle(struct EnuCoor_i *wp_center, int32_t radius
    )
{
  ...

  // direction of rotation
  int8_t sign_radius = radius > 0 ? 1 : -1;
  // absolute radius
  int32_t abs_radius = abs(radius);
  // carrot_angle
  int32_t carrot_angle = ((CARROT_DIST << INT32_ANGLE_FRAC)
     / abs_radius);
  Bound(carrot_angle, (INT32_ANGLE_PI / 16),
     INT32_ANGLE_PI_4);
  carrot_angle = nav_circle_qdr - sign_radius *
     carrot_angle;

  ...
}
```

Listing 5: Safe Heading Change in Guidance [15]

4) Safe Leg Distance in Guidance: (a) Use Context: For linear trajectories that do not involve heading change, Paparazzi also uses carrot-based guidance. In this setting, the distance between the starting point of the leg and the carrot, which is called *leg distance*, is dynamically adjusted. (b) Datatype: distance (X and Y axis). (c) The Need for BFs: if the leg distance is set too long, the UAV may go "past" the waypoint of the target point. Deviating from the planned trajectory is a correctness concern. (d) Example: In this code snippet [16] which concerns route (i.e., linear) navigation, the `nav_leg_progress` is bounded to guarantee that the next leg of flight does not surpass the target waypoint. (e) Occurrence: 4 instances.

E. Sensor Management

As important components of a UAV, sensors play an irreplaceable role in UAV's state estimation, e.g., UAV's current attitude (pitch/roll/yaw). An accurate estimation based on sensor data is also critical for UAV safety. The physical variables related to sensor management are *sensor readings*, the *time interval* among readings, and the *weight* when multiple sensor readings are weighted. We identified instances of BFs applied for sensor management, which we divide into 3 use scenarios.

1) Safe Sensor Readings: (a) Use Context: The raw sensing data may be unreliable, either because the sensor is faulty, or because the reading may only reflect a transient state. (b) Datatype: sensor reading. (c) The Need for BFs: The need for bounding is sensor-specific. Take the current sensor for example. Due to overflow on high current spikes (fast electrical transients in current), the reading may be magnitudes higher than normal readings. This would impact battery estimation, crucial for estimating the remaining flight time. (d) Example: In this code snippet [17] , the current sensor keeps its readings in `electrical.current`, which is in turn bound to a safe range $[-65000, 65000]$. (e) Occurrence: 1 instance.

2) Safe Sensor Reading Interval: (a) Use Context: In UAVs, sensors are continuously read. In some scenarios, the time interval between different readings plays a crucial role in physical estimation. For example, as an application of Kalman filter [5], the UAV can use data from GPS and barometer at different time intervals to estimate its vertical position and velocity. (b) Datatype: time interval. (c) The Need for BFs: if there is a significant delay between two intervals, the estimation may be inaccurate, which in turn severely impacts the decision-making process of the UAV. (d) Example: In this code snippet [18] , the variable `dt` represents the time interval between two GPS readings. It is bounded into the range $[0.02, 2]$ seconds. The variable is used by Kalman filter (`alt_kalman`) for the estimation of the UAV's altitude and vertical speed. (e) Occurrence: 1 instance.

3) Safe Sensor Fusion: (a) Use Context: Complementary filter [6] combines sensor readings from the accelerometer and the gyroscope to estimate UAV attitude (pitch/roll/yaw). (b) Datatype: weight for sensor fusion (c) The Need for BFs:

[15]Function `nav_circle` at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/firmwares/rotorcraft/navigation.c

[16]Function `nav_route` at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/firmwares/rotorcraft/navigation.c

[17]Function `electrical_periodic` at https://github.com/paparazzi/paparazzi/blob/363dec86938cd1090221ccd772fc6fae58ed89a2/sw/airborne/subsystems/electrical.c

[18]Function `ins_alt_float_update_gps` at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/subsystems/ins/ins_alt_float.c

To ensure that data collected from both sensors are considered adequately, their proportions in attitude estimation need bounding in order to reach a balance between these two components. (d) Example: In Listing 6, `ahrs_fc.weight` computed at line 9 reflects the role of accelerometer plays in attitude estimation, which is influenced by `fabs(1.0 - g_meas_norm)`, the deviation between the measured gravitational acceleration and 1g. In the case of vibrations, large deviations from 1g may cause a decrement of the weight for the accelerometer data if bound is not introduced, ultimately causing the attitude estimate to drift [7]. Attitude estimation is critical for the safety of UAVs. In the aviation history, a catastrophe with the same root cause is Lion Air Flight 610, which was caused by incorrect angle-of-attack sensing (and consequent activation of the anti-stall software to repeatedly pitch the plane downward) [8]. (e) Occurrence: 6 instances.

```
1  void ahrs_fc_update_accel(struct FloatVect3 *accel, float
       dt)
2  {
3    ...
4
5    // compute ratio between measured gravitational
       acceleration and the standard value
6    const float g_meas_norm = float_vect3_norm(&
       filtered_gravity_measurement) / 9.81;
7
8    // compute the weight of accelerometer in attitude
       estimation
9    ahrs_fc.weight = 1.0 - ahrs_fc.gravity_heuristic_factor *
        fabs(1.0 - g_meas_norm) / 10.0;
10
11   Bound(ahrs_fc.weight, 0.15, 1.0);
12
13   ...
14 }
```

Listing 6: An Example of Sensor Fusion [19]

References

[1] H. W. Ho, G. C. H. E. de Croon, E. van Kampen, Q. P. Chu, and M. Mulder, "Adaptive gain control strategy for constant optical flow divergence landing," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 508–516, April 2018.

[2] Paparazzi Wiki. (2017) Rotorcraft configuration. [Online; accessed 27-February-2020]. [Online]. Available: http://wiki.paparazziuav.org/wiki/Rotorcraft_Configuration#Motor_Mixing

[3] A. Budiyanto, A. Cahyadi, T. B. Adji, and O. Wahyunggoro, "Uav obstacle avoidance using potential field under dynamic environment," in *2015 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, Bandung, 2015, pp. 187–192.

[4] G. Conte, S. Duranti, and T. Merz, "Dynamic 3d path following for an autonomous helicopter," in *Proceedings of the 5th IFAC Symposium on Intelligent Autonomous Vehicles*, Oxford, UK, 2004, pp. 473–478.

[5] R. E. Kalman, "A new approach to linear filtering and prediction problems" transaction of the asme journal of basic," 1960.

[6] W. T. Higgins, "A comparison of complementary and kalman filtering," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-11, no. 3, pp. 321–325, May 1975.

[7] Paparazzi Wiki. (2015) Vibration. [Online; accessed 10-February-2020]. [Online]. Available: https://wiki.paparazziuav.org/wiki/Vibration#Complementary_AHRS

[8] D. Shortell and J. Shelley, "Lion air crash investigators looking at two american companies associated with boeing 737 max sensor," *CNN*, Apr 2019. [Online]. Available: https://www.cnn.com/2019/04/04/us/boeing-sensor-investigation/index.html

[19]Function `ahrs_fc_update_accel` at https://github.com/paparazzi/paparazzi/blob/master/sw/airborne/subsystems/ahrs/ahrs_float_cmpl.c