Model-based Engineering for Quadcopter Control Software

VRUSHANK AGRAWAL

ADVISOR: TIMOTHY BOURKE

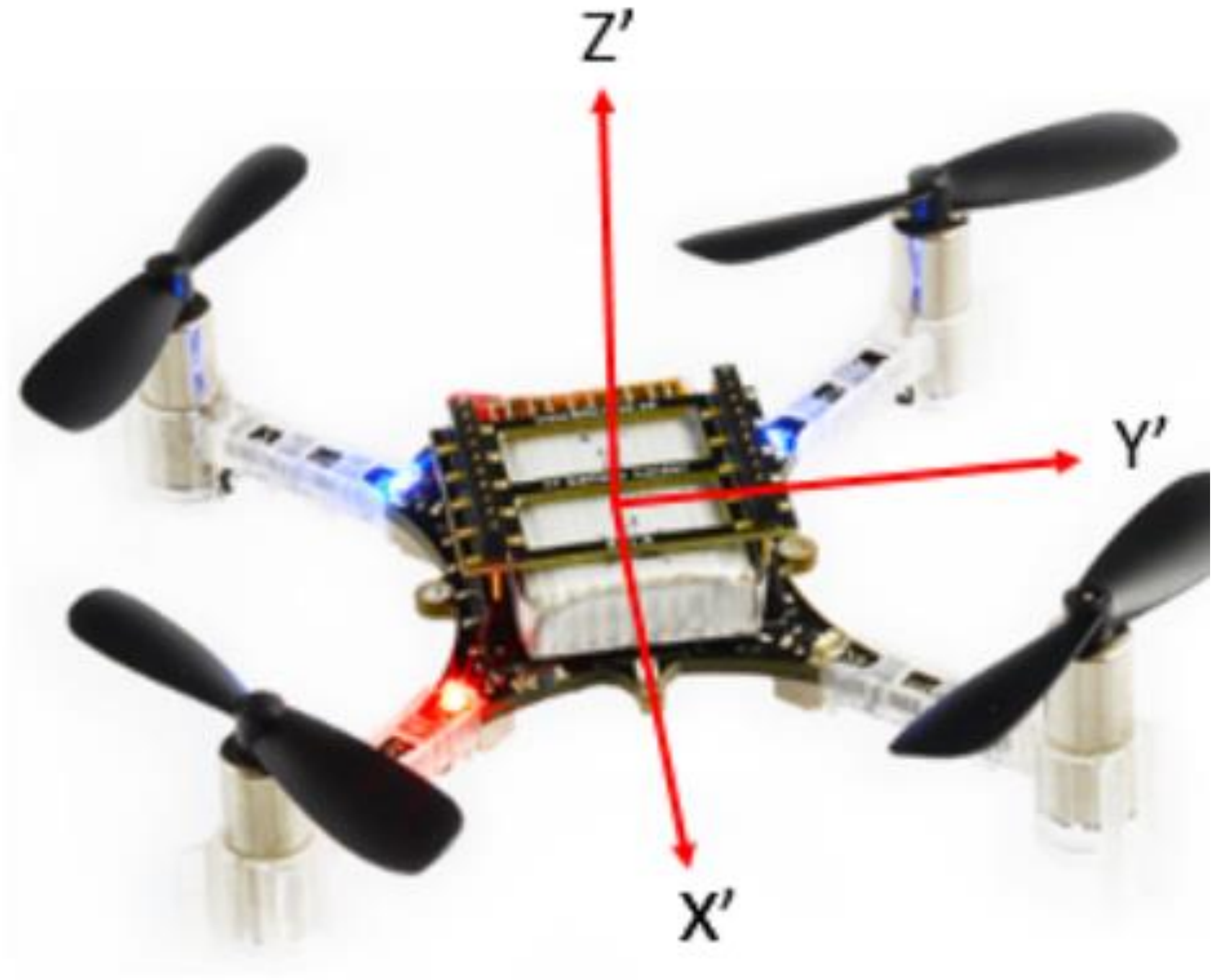
CO-*ADVISOR:* BASILE PESIN

Introduction

- Quadcopter
  - Unmanned Aerial Vehicle and a type of drone
  - Embedded System which uses
    - RTOS
    - parallelism
    - works with time constraints

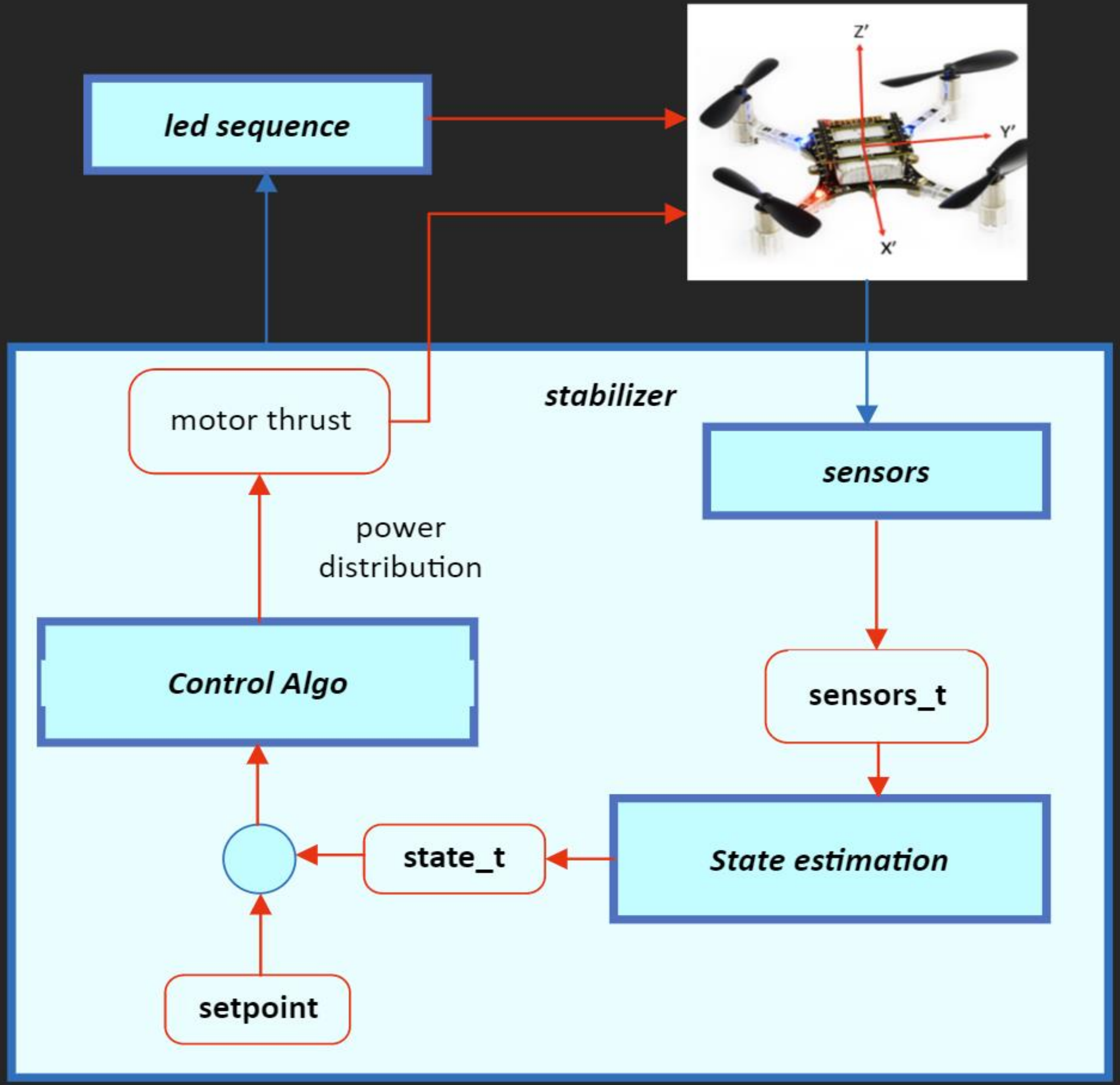
Quadcopter Movement

- 6 Degrees of Freedom (DOF)
- 4 State Variables
  - Thrust
  - Roll
  - Pitch
  - Yaw



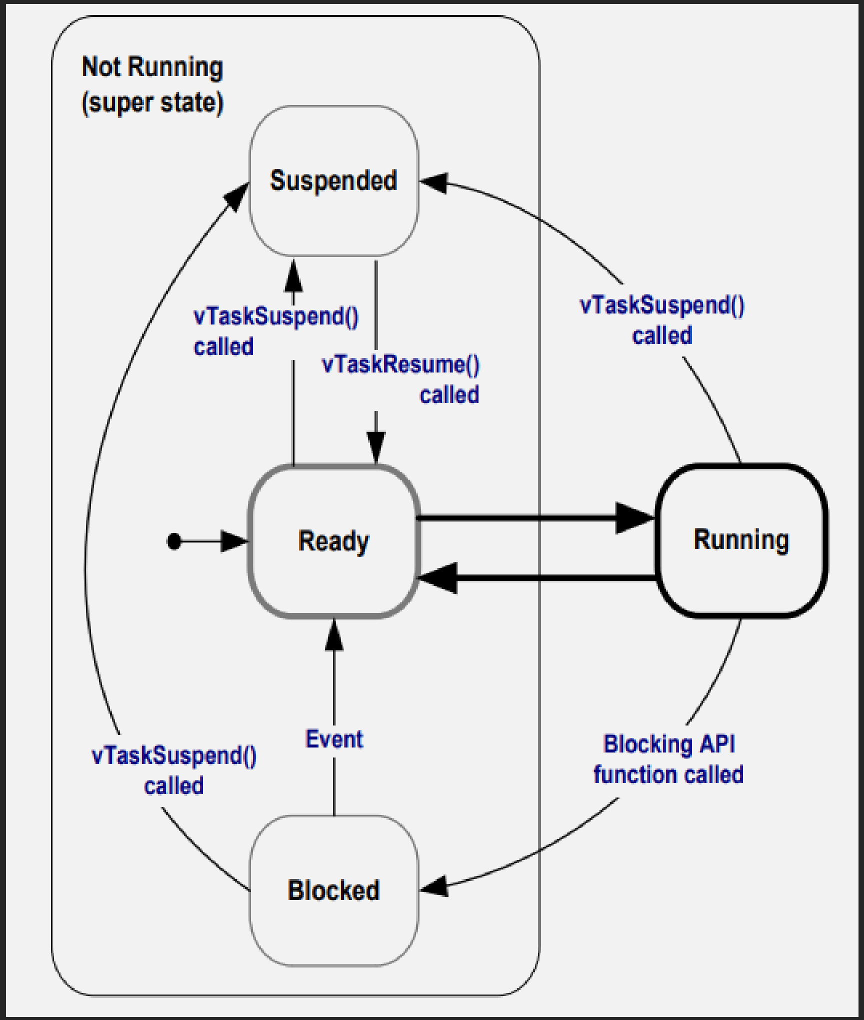
Crazyflie system design

- A combination of tasks that run parallelly on the microcontroller
- Led sequence for setting the leds
- Feedback control loop
  - Takes sensors' input
  - Gives controlled input to the hardware



FreeRTOS Scheduler

- FreeRTOS
- Task states in FreeRTOS
- Preemptive scheduler
- Idle Task



Problem

- Quadcopters written in C/C++ from initial state diagrams
- Hard to understand the code and model and test system behavior

```c
static void stabilizerTask(void* param)
{
  uint32_t tick;
  uint32_t lastWakeTime;
  vTaskSetApplicationTaskTag(0, (void*)TASK_STABILIZER_ID_NBR);

  //Wait for the system to be fully started to start stabilization loop
  systemWaitStart();

  DEBUG_PRINT("Wait for sensor calibration...\n");

  // Wait for sensors to be calibrated
  lastWakeTime = xTaskGetTickCount();
  while(!sensorsAreCalibrated()) {
    vTaskDelayUntil(&lastWakeTime, F2T(RATE_MAIN_LOOP));
  }
  // Initialize tick to something else then 0
  tick = 1;

  rateSupervisorInit(&rateSupervisorContext, xTaskGetTickCount(), M2T(1000), 997, 1003, 1);

  DEBUG_PRINT("Ready to fly.\n");

  while(1) {
    // The sensor should unlock at 1kHz
    sensorsWaitDataReady();

    // update sensorData struct (for logging variables)
    sensorsAcquire(&sensorData, tick);

    if (healthShallWeRunTest()) {
      healthRunTests(&sensorData);
    } else {
      // allow to update estimator dynamically
      if (getStateEstimator() != estimatorType) {
```

Alternative Approach

- Write the code in a Dataflow Synchronous Language like Heptagon
- Inspired from Lustre and offers modern features like *delays, automata, switch*

```
node everyn(n : int) returns (v : bool);
var x : int;
let
  x = 0 fby ((x + 1) % n);
  v = (x = 0);
tel
```

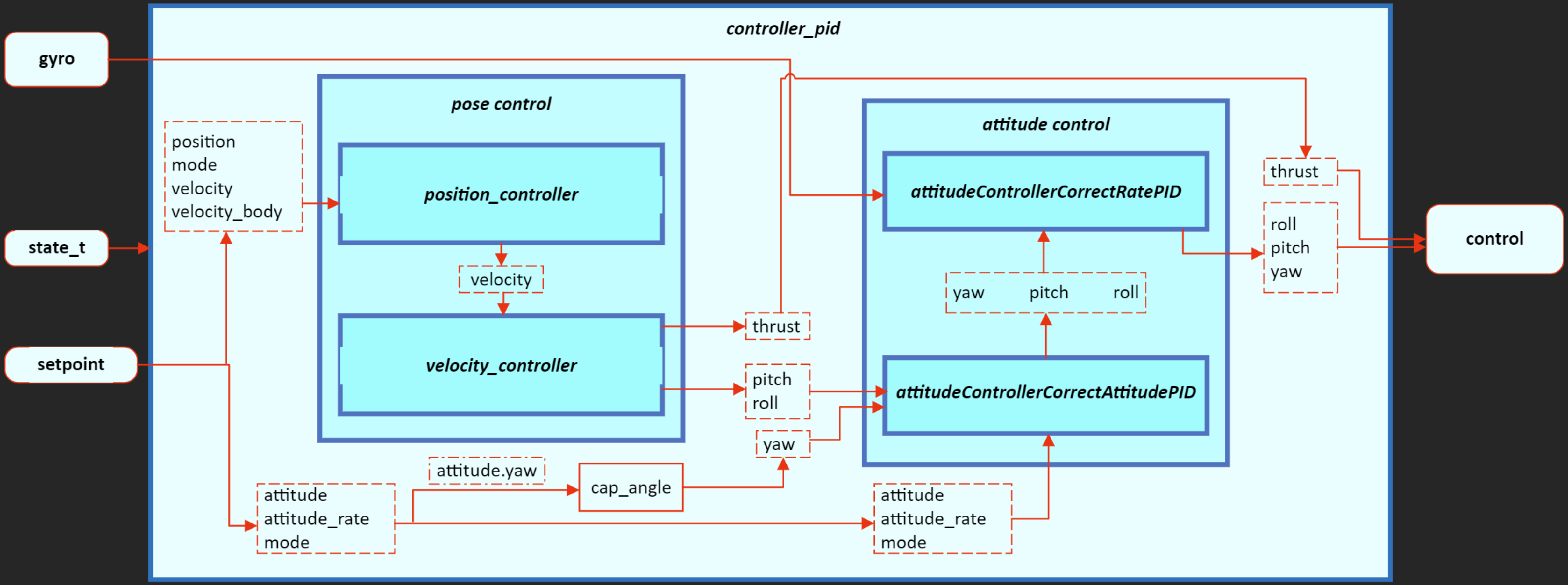
```
node led_on_every_n(n : int) returns(led_state : bool);
var timer : int;
let
    timer = (0 fby (timer+1)) % n;
    automaton
        state ON
            do led_state = true
            unless (timer > 0) then OFF
        state OFF
            do led_state = false
            unless (timer = 0) then ON
    end;
tel
```

```
type action = Change | Previous

node ledseq_task(led_action: action) returns(last led_state : bool = false);
let
    switch (led_action)
    | Change do led_state = led_on_every_n(2)
    | Previous do led_state = last led_state
    end;
tel
```

Our Goal

- Write parts of the Crazyflie source code (written in **C**) in Heptagon
- The thesis is in continuation with CSE303 project which was dedicated to understanding the Crazyflie source code
- PID controller already implemented by colleagues, so, integrate and test the integration in Crazyflie.
- Implement state estimation, led sequence task, and Kalman task in Heptagon

Cascaded PID Controller Dataflow diagram



Merging Heptagon-generated C code

- For every node, Heptagon generates two data types:
  - *mem* - contains reference to the current state
  - *out* - contains output of node.

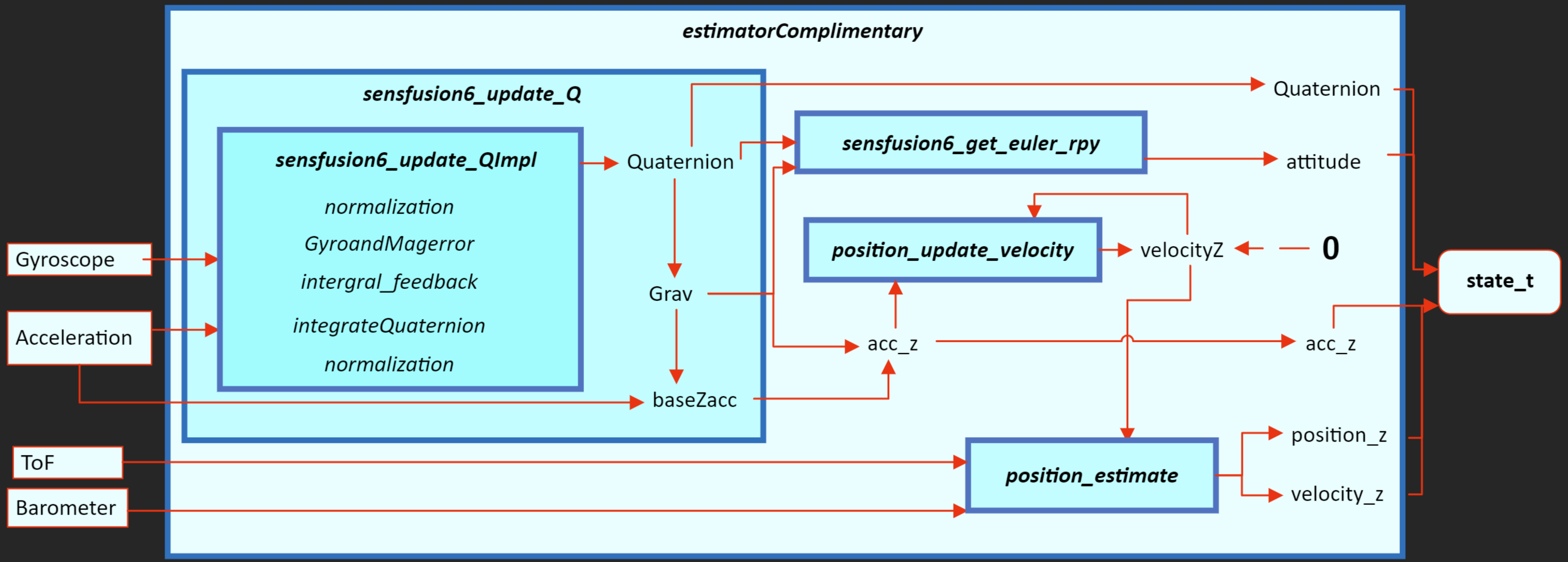
- And two types of functions :
  - *reset* – initializes temporary variables used in the node
  - *step* – evaluates code of the Heptagon node

```
node foo(input: float)
returns (output: float);
var last temp: float = 0.0;
let
    temp = input;
    output = temp;
tel
```

```
typedef struct Libel__foo_mem {
  float temp_1;
} Libel__foo_mem;

typedef struct Libel__foo_out {
  float output;
} Libel__foo_out;
```

```
void Libel__foo_reset(Libel__foo_mem* self) {
  self->temp_1 = 0.000000f;
}

void Libel__foo_step(float input,
                     Libel__foo_out* _out,
                     Libel__foo_mem* self) {
  _out->output = input;
  self->temp_1 = input;
}
```

Complementary Filter Dataflow diagram



Complementary Filter Debugging

- Debug testing environment

- Bug in Heptagon compiler

- Fast inverse square root and Floating-point numbers

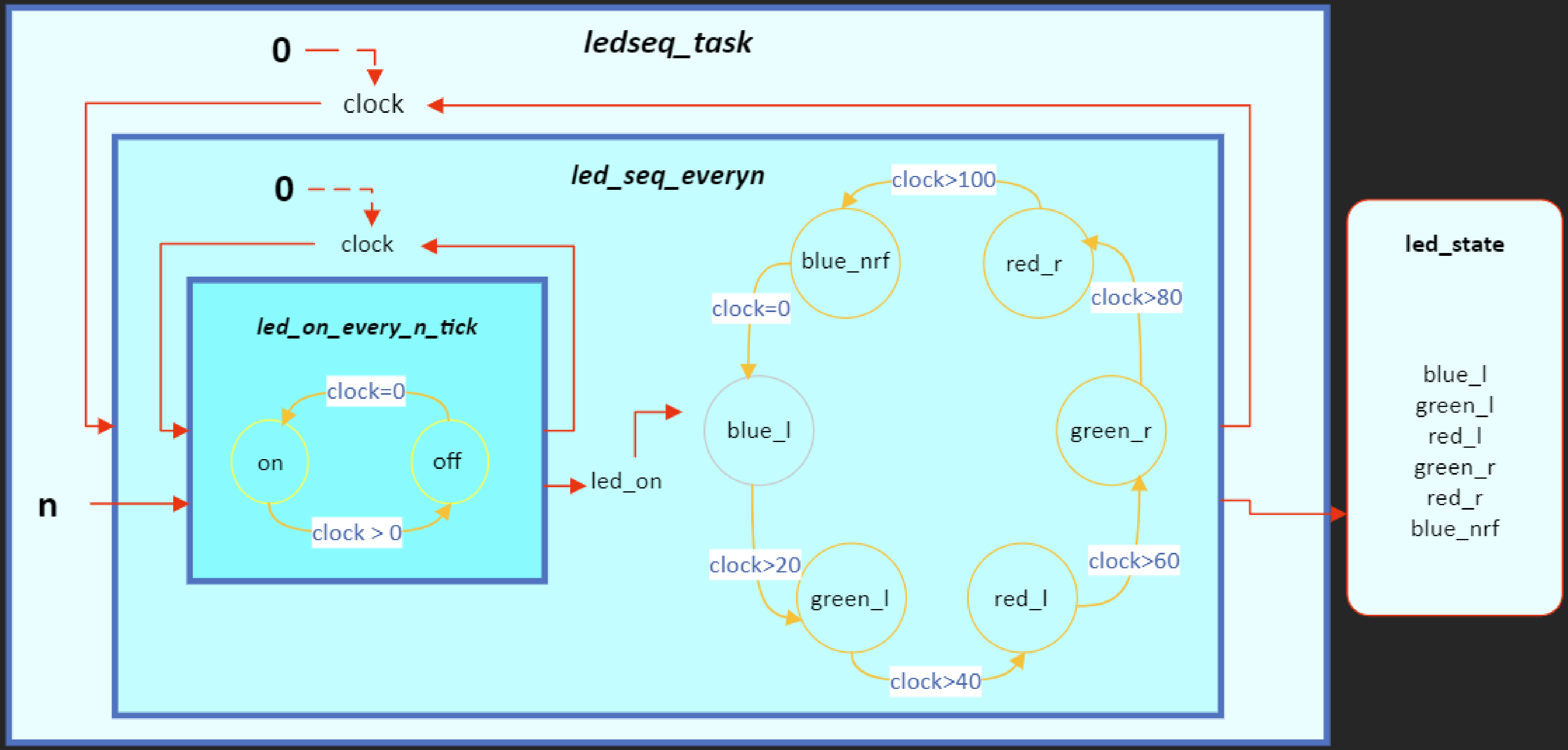
```
node estimator_complementary(sensor: sensor_data_est)
returns (st : state_t);

... (See appendix D)

    last st_attitude_this: attitude = {roll = 0.0; pitch = 0.0; yaw = 0.0};
    last st_attitude_quat_this: quaternion = {qw = 1.0; qx = 0.0; qy = 0.0; qz = 0.0};
    last st_acc_z: float = 0.0;
    last st_position_z: float = 0.0;
    last st_velocity_z: float = 0.0;

... (See appendix D)

    st = {  st_attitude = st_attitude_this;
            st_attitude_quat = st_attitude_quat_this;
            st_position = {x = 0.0; y = 0.0; z = st_position_z};
            st_velocity = {x = 0.0; y = 0.0; z = st_velocity_z};
            st_acc = {x = 0.0; y = 0.0; z = st_acc_z}};
tel
```

```
reference: recipNorm: -1.000000
reference: qw: -0x1.fc12p-1, qx: -0x1.adb2e4p-8, qy: 0x1.4ba74ap-10, qz: -0x1.be57a4p-4
Heptagon: recipNorm: 1.000000
Heptagon: qw: -0x1.fc12p-1 qx: -0x1.adb2e4p-8 qy: 0x1.4ba76ep-10 qz: -0x1.be57a6p-4
```

```
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

    return y;
}
```

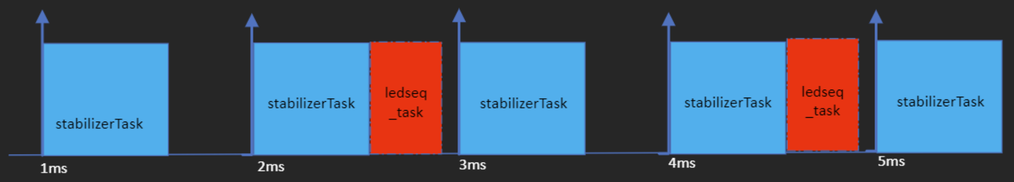
Led Seq Dataflow diagram

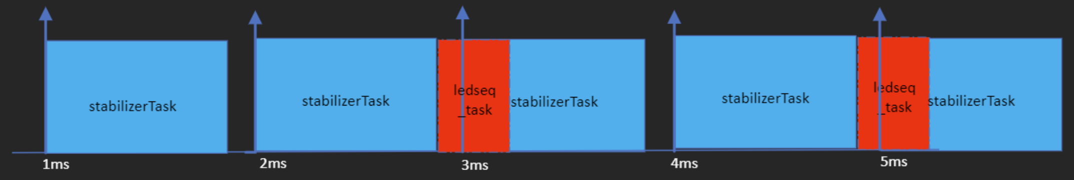
- Final output
- Internal timers
- State machines

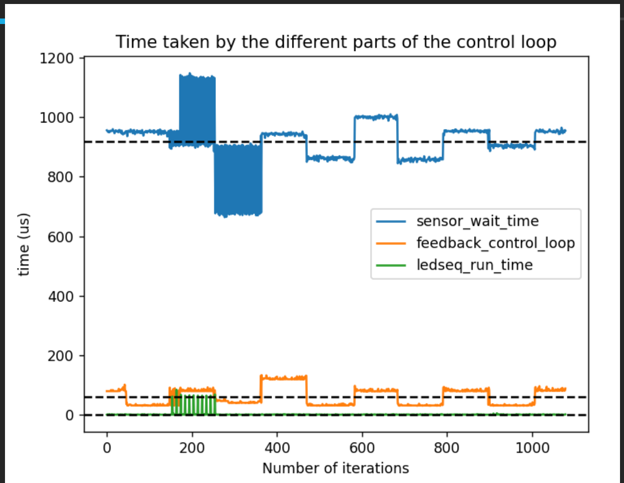


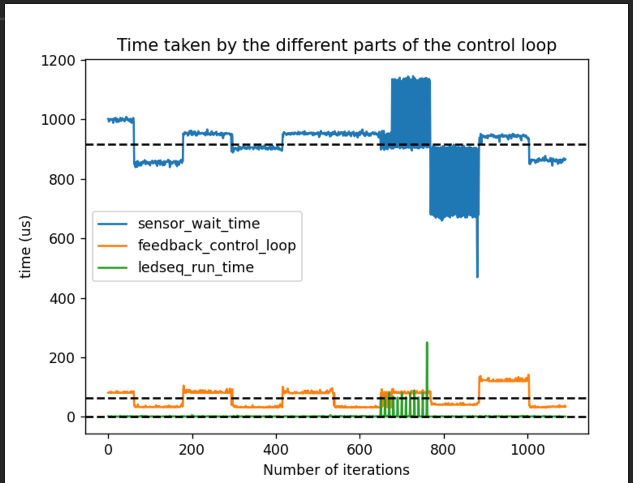
Led sequence FreeRTOS task analysis

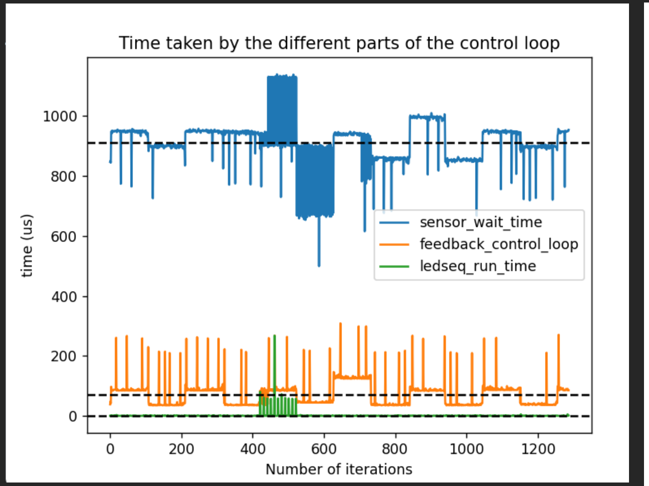
- Possible task running time can vary as show in the diagram
- Timing aspect analysis

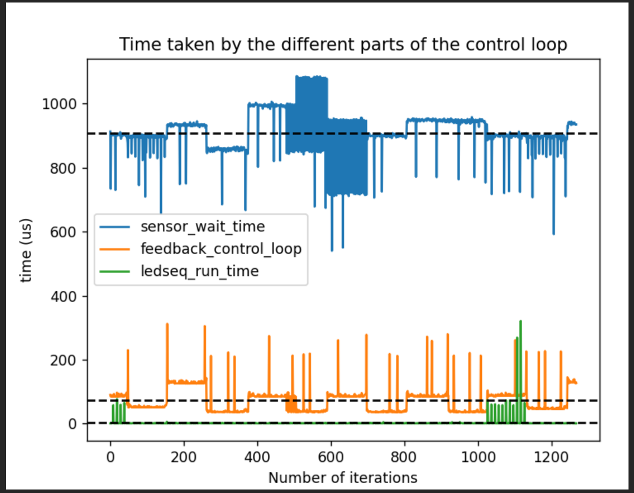












Extended Kalman Filter

- Rearranging code

- Issues with stack pointer debugged using disassembly

```c
static void kalmanTask(void* parameters) {…

void estimatorKalman(state_t *state, const uint32_t tick)
{
  // This function is called from the stabilizer loop. It is important that this call returns
  // as quickly as possible. The dataMutex must only be locked short periods by the task.
  // xSemaphoreTake(dataMutex, portMAX_DELAY);

  uint32_t osTick = xTaskGetTickCount(); // would be nice if this had a precision higher than 1ms...
  updateQueuedMeasurements(osTick);

  libel_from_axis3f_kalman(&accAccumulator, &acc_accumulator);
  libel_from_axis3f_kalman(&gyroAccumulator, &gyro_accumulator);
  libel_from_axis3f_kalman(&accLatest, &acc_latest);
  libel_from_coreData(&coreData, &kalman_coredata);

  Libel__kalman_task_step(
    kalman_coredata,
    (float)gyroAccumulatorCount, (float)accAccumulatorCount,
    gyro_accumulator,
    acc_accumulator, doneUpdate,
    acc_latest,
    &kalman_task_out, &kalman_task_mem
  );
```

```
node kalman_task(core_data: kalman_coredata_t;
                 gyro_acc_count, acc_acc_count: float;
                 gyro_accumulator, acc_accumulator: vec3;
                 done_update: bool;
                 acc_latest: vec3)
returns (st : state_t; core_data_3: kalman_coredata_t;
         gyro_acc_count_updated, acc_acc_count_updated: float;
         gyro_accumulator_updated, acc_accumulator_updated: vec3);
var …
let
    prediction_freq = everyn<<10>>();

    switch prediction_freq
    | true do
        (core_data_0, acc_acc_count_temp, gyro_acc_count_temp, gyro_accumulator_temp, acc_accumulator_temp) =
        predict_state_forward(core_data, gyro_acc_count, acc_acc_count, gyro_accumulator, acc_accumulator);
    | false do
        core_data_0 = core_data;
    end;

    core_data_1 = kalman_core_add_process_noise(core_data);

    switch done_update
    | true do
        core_data_2 = kalman_core_finalize(core_data);
        reset_estimation = not kalman_supervisor_is_state_within_bounds(core_data_2);
    | false do
        core_data_2 = core_data;
    end;

    st = kalman_core_externalize_state(core_data_2, acc_latest);

    switch not (reset_estimation)
    | true do…
    | false do…
    end;
```

```
00000000 <estimatorKalman>:
   0: e92d 4ff0     stmdb sp!, {r4, r5, r6, r7, r8,
   4: ed2d 8b0c     vpush {d8-d13}
   8: b0f7          sub    sp, #476 ; 0x1dc
   a: 4681          mov    r9, r0
   c: f7ff fffe     bl 0 <xTaskGetTickCount>
  10: 4c9a          ldr    r4, [pc, #616] ; (27c <est
  12: 4d9b          ldr    r5, [pc, #620] ; (280 <est
```

```
00000000 <stabilizerTask>:
   0: e92d 41f0     stmdb sp!, {r4, r5, r6, r
   4: ed2d 8b02     vpush {d8}
   8: 2103          movs   r1, #3
   a: b088          sub    sp, #56
   c: 2000          movs   r0, #0
   e: f7ff fffe     bl 0 <vTaskSetApplication
  12: f7ff fffe     bl 0 <systemWaitStart>
```

Conclusion

Results

Parts of feedback control loop and FreeRTOS tasks (led sequence) were successfully implemented as Heptagon nodes

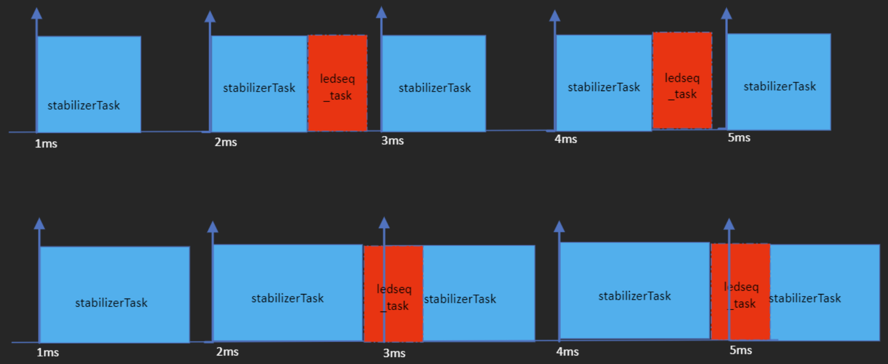
Dataflow diagrams from Heptagon can be used to generate the low-level C code

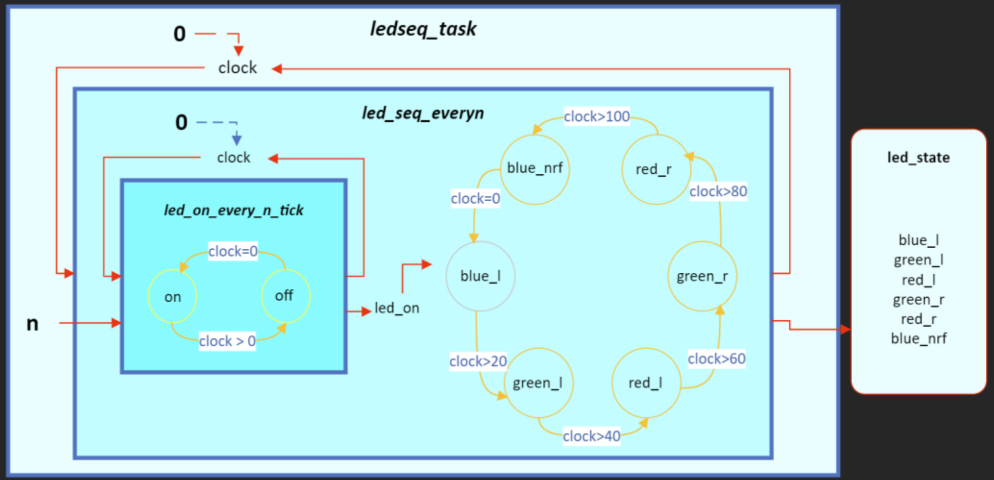
Further Steps

Extended Kalman Filter can be debugged and integrated in the firmware.

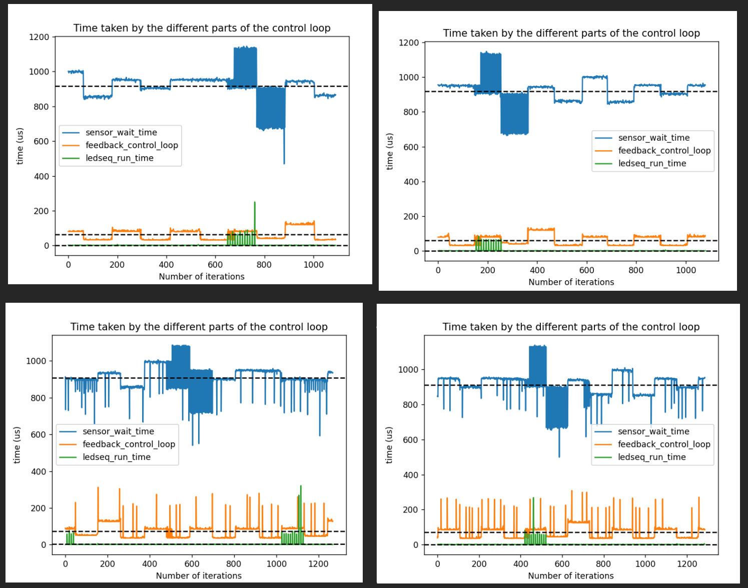
Other FreeRTOS tasks including the High-Level Commander can be integrated as Heptagon nodes.

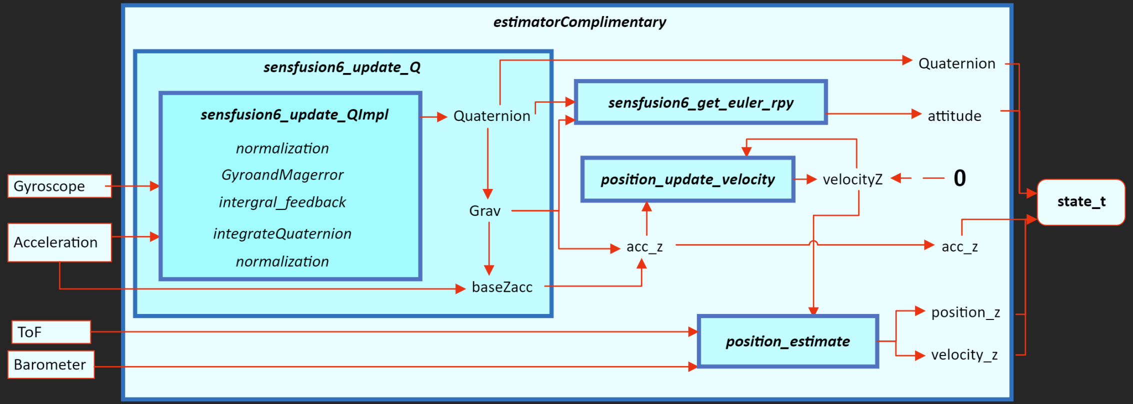
```c
stateEstimator(&state, tick);
compressState();

if (crtpCommanderHighLevelGetSetpoint(&tempSetpoint, &state, tick)) {
  commanderSetSetpoint(&tempSetpoint, COMMANDER_PRIORITY_HIGHLEVEL);
}

commanderGetSetpoint(&setpoint, &state);
compressSetpoint();

collisionAvoidanceUpdateSetpoint(&setpoint, &sensorData, &state, tick);

controller(&control, &setpoint, &sensorData, &state, tick);

checkEmergencyStopTimeout();
```



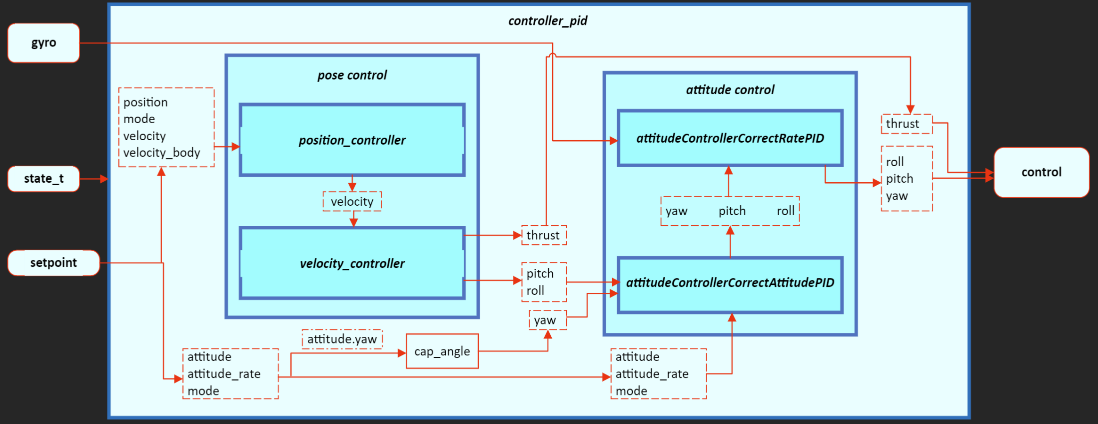



```
node estimator_complementary(sensor: sensor_data_est)
returns (st : state_t);

... (See appendix D)

    last st_attitude_this: attitude = {roll = 0.0; pitch = 0.0; yaw = 0.0};
    last st_attitude_quat_this: quaternion = {qw = 1.0; qx = 0.0; qy = 0.0; qz = 0.0};
    last st_acc_z: float = 0.0;
    last st_position_z: float = 0.0;
    last st_velocity_z: float = 0.0;

... (See appendix D)

    st = {  st_attitude = st_attitude_this;
            st_attitude_quat = st_attitude_quat_this;
            st_position = {x = 0.0; y = 0.0; z = st_position_z};
            st_velocity = {x = 0.0; y = 0.0; z = st_velocity_z};
            st_acc = {x = 0.0; y = 0.0; z = st_acc_z}};
tel
```






```c
float Q_rsqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;                       // evil floating point bit level hacking
    i  = 0x5f3759df - ( i >> 1 );               // what the fuck?
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );   // 1st iteration
//  y  = y * ( threehalfs - ( x2 * y * y ) );   // 2nd iteration, this can be removed

    return y;
}
```



Thank You