pozyx-shield.ino

```
#include <Pozyx.h>
#include <Pozyx_definitions.h>

#include <SoftwareSerial.h>
#include <Wire.h>

////////////////// Pozyx Prams ////////////////////////////////
#define CONFIG_TX_GAIN 33.0f

#define NUM_ANCHORS 4
// the network id of the anchors: change these to the network ids of
your anchors.
uint16_t anchor_id[4] = { 0x610e, // (0,0)
                          0x610f, // x-axis
                          0x6166, // y-axis
                          0x6167};

// only required for manual anchor calibration.
// Please change this to the coordinates measured for the anchors
int32_t anchors_x[NUM_ANCHORS] = {0,     4700, 0,     4700};    //
anchor x-coorindates in mm (horizontal)
int32_t anchors_y[NUM_ANCHORS] = {0,     0,     5150, 5150};    //
anchor y-coordinates in mm (vertical)
int32_t heights[NUM_ANCHORS] =   {-1400, -1450, -1350, -1550};    //
anchor z-coordinates in mm (1.2m above vehicle's starting altitude)

// RX TX serial for flight controller ex) Pixhawk
// https://github.com/PaulStoffregen/AltSoftSerial
SoftwareSerial fcboardSerial(10, 11); // rx, tx

#define MSG_HEADER          0x01
#define MSGID_BEACON_CONFIG 0x02
#define MSGID_BEACON_DIST   0x03
#define MSGID_POSITION      0x04

// structure for messages uploaded to ardupilot
union beacon_config_msg {
    struct {
        uint8_t beacon_id;
        uint8_t beacon_count;
        int32_t x;
        int32_t y;
        int32_t z;
    } info;
    uint8_t buf[14];
};
union beacon_distance_msg {
    struct {
        uint8_t beacon_id;
```

```
        uint32_t distance;
    } info;
    uint8_t buf[5];
};
union vehicle_position_msg {
    struct {
        int32_t x;
        int32_t y;
        int32_t z;
        int16_t position_error;
    } info;
    uint8_t buf[14];
};
////////////////////////////////////////////////

void setup()
{
    Serial.begin(115200);
    fcboardSerial.begin(115200);

    if (Pozyx.begin() == POZYX_FAILURE) {
        Serial.println(("ERR: shield"));
        delay(100);
        abort();
    }

    Serial.println(("V1.0"));

    // clear all previous devices in the device list
    Pozyx.clearDevices();

    // configure beacons
    while (!configure_beacons()) {
        delay(1000);
    }

    int status = Pozyx.doAnchorCalibration(POZYX_2_5D, 10, NUM_ANCHORS,
anchor_id, heights);
    if (status != POZYX_SUCCESS) {
      Serial.println(status);
      Serial.println(("ERROR: calibration"));
      Serial.println(("Reset required"));
      delay(100);
      abort();
    }

    // if the automatic anchor calibration is unsuccessful, try
manually setting the anchor coordinates.
    // fot this, you must update the arrays anchors_x, anchors_y and
```

```
heights above
    // comment out the doAnchorCalibration block and the if-statement
above if you are using manual mode
    // SetAnchorsManual();

    print_anchor_coordinates();

    Serial.println(("Waiting.."));
    delay(5000);

    Serial.println(("Starting: "));
}

void loop()
{
    static uint32_t loop_start = 0;
    static uint8_t stage = 0;   // 0 = initialisation, 1 = normal
flight
    static uint16_t beacon_sent_count = 0;
    static uint32_t beacon_sent_time = 0;

    // initialise start time
    if (loop_start == 0) {
        loop_start = millis();
    }

    // advance to normal flight stage after 1min
    if (stage == 0) {
        uint32_t time_diff =  (millis() - loop_start);
        if (time_diff > 60000) {
            stage = 1;
            Serial.println("Stage1");
        }
    }

    // slow down counter
    static uint8_t counter = 0;
    counter++;
    if (counter >= 20) {
        counter = 0;
    }

    // during stage 0 (init) send position and beacon config as quickly
as possible
    // during stage 1 send about every 2 seconds
    if (stage == 0 || counter == 0) {
        send_beacon_config();
        get_position();
        if (beacon_sent_count > 0 && beacon_sent_time != 0) {
            uint32_t time_diff = millis() - beacon_sent_time;
            float hz = (float)beacon_sent_count / (time_diff /
```

```
1000.0f);
            Serial.print("Beacon hz:");
            Serial.println(hz);
        }
        beacon_sent_count = 0;
        beacon_sent_time = millis();
    }

    // send beacon distances
    get_ranges();
    beacon_sent_count++;
}

uint32_t time_start_ms;
void timer_start()
{
    time_start_ms = millis();
}
void timer_end()
{
    uint32_t time_diff = millis() - time_start_ms;
    Serial.print("ms:");
    Serial.println(time_diff);
}

void print_comma()
{
    Serial.print(",");
}

void print_tab()
{
    Serial.print("\t");
}

// set a tag or anchor's gain
//   set tag_id to zero to set local device's gain
//   returns true on success
bool set_device_gain(uint16_t dev_id, float gain)
{
    float tx_power = -1;

    // get/set transmit power of tag
    bool gain_ok = false;
    uint8_t retry = 0;
    while (!gain_ok && retry < 5) {
        if (Pozyx.getTxPower(&tx_power, dev_id) == POZYX_SUCCESS) {
            if (tx_power != gain) {
                Pozyx.setTxPower(CONFIG_TX_GAIN, dev_id);
```

```
            } else {
                gain_ok = true;
            }
        }
        retry++;
    }

    // display final gain
    Serial.print("Dev ");
    Serial.print(dev_id, HEX);
    Serial.print(" gain ");
    if (tx_power > 0) {
        Serial.print(tx_power);
    } else {
        Serial.print("unknown");
    }
    Serial.print(" (retry ");
    Serial.print(retry);
    Serial.print(")");
    Serial.println();

    return gain_ok;
}

// performs repeated calls to get reliable distance between devices
bool get_remote_range(uint16_t dev1, uint16_t dev2, int32_t& range)
{
    // set distances between tags
    uint32_t range_tot = 0;
    uint16_t count = 0;
    device_range_t dev_range;
    for (uint8_t i=0; i <= 10; i++) {
        // origin to 1st
        if (Pozyx.doRemoteRanging(dev1, dev2, &dev_range) ==
POZYX_SUCCESS) {
            range_tot += dev_range.distance;
            count++;
        }
        if (Pozyx.doRemoteRanging(dev2, dev1, &dev_range) ==
POZYX_SUCCESS) {
            range_tot += dev_range.distance;
            count++;
        }
    }
    // success if at least 5 successful ranges were retrieved
    if (count > 5) {
        range = range_tot / count;
        return true;
    }
    return false;
}
```

```
void print_failed_to_range(uint16_t dev1, uint16_t dev2)
{
    Serial.print("ranging fail ");
    Serial.print(dev1,HEX);
    Serial.print(" to ");
    Serial.println(dev2,HEX);
}

void set_beacon_position(uint8_t index, int32_t x_mm, int32_t y_mm,
int32_t z_mm)
{
    anchors_x[index] = x_mm;
    anchors_y[index] = y_mm;
    heights[index] = z_mm;
}

// configure beacons
bool configure_beacons()
{
    bool configured_ok = true;

    // get/set transmit power of tag
    if (!set_device_gain(0, CONFIG_TX_GAIN)) {
        configured_ok = false;
    }

    // set transmit power of beacons
    for (uint8_t i=0; i < NUM_ANCHORS; i++) {
        if (!set_device_gain(anchor_id[i], CONFIG_TX_GAIN)) {
            configured_ok = false;
        }
    }

    // set distances between tags
    int32_t x_range = 0, y_range = 0;
    // origin to x-axis (i.e. bottom right)
    if (get_remote_range(anchor_id[0], anchor_id[1], x_range)) {
        set_beacon_position(1, x_range, 0, heights[1]);
    } else {
        print_failed_to_range(anchor_id[0], anchor_id[1]);
        configured_ok = false;
    }
    // origin to y-axis (i.e. top left)
    if (get_remote_range(anchor_id[0], anchor_id[2], y_range)) {
        set_beacon_position(2, 0, y_range, heights[2]);
    } else {
        print_failed_to_range(anchor_id[0], anchor_id[2]);
        configured_ok = false;
```

```
    }
    // top right
    if (x_range != 0 && y_range != 0) {
        set_beacon_position(3, x_range, y_range, heights[3]);
    } else {
        Serial.println("beacons too close");
        configured_ok = false;
    }

    if (configured_ok) {
        Serial.println("Beacon Configuration complete");
    } else {
        Serial.println("Beacon Configuration failed!");
    }

    return configured_ok;
}

// function to manually set the anchor coordinates
void SetAnchorsManual()
{
    for (uint8_t i=0; i<NUM_ANCHORS; i++) {
        device_coordinates_t anchor;
        anchor.network_id = anchor_id[i];
        anchor.flag = 0x1;
        anchor.pos.x = anchors_x[i];
        anchor.pos.y = anchors_y[i];
        anchor.pos.z = heights[i];
        Pozyx.addDevice(anchor);
    }
}

// print coordinates to the serial monitor
void print_coordinates(coordinates_t coor, pos_error_t pos_error)
{
    Serial.print("Pos x:");
    Serial.print(coor.x);
    print_tab();
    Serial.print("y:");
    Serial.print(coor.y);
    print_tab();
    Serial.print("z:");
    Serial.print(coor.z);
    Serial.print(" err x:");
    Serial.print(pos_error.x);
    Serial.print(" y:");
    Serial.print(pos_error.y);
    Serial.println();
}

// print out the anchor coordinates
```

```
void print_anchor_coordinates()
{
  uint8_t list_size;
  int status;

  status = Pozyx.getDeviceListSize(&list_size);
  Serial.print("list: ");
  Serial.println(status*list_size);

  // print error if no anchors are setup
  if (list_size == 0) {
    Serial.println("No Anchors");
    Serial.println(Pozyx.getSystemError());
    return;
  }

  // retrieve anchor information
  uint16_t device_ids[list_size];
  status &= Pozyx.getDeviceIds(device_ids,list_size);

  Serial.print("Anchors found: ");
  Serial.println(list_size);

  coordinates_t anchor_coor;

  for (int i=0; i<list_size; i++) {
    Serial.print("A0x");
    Serial.print(device_ids[i], HEX);
    print_comma();
    status = Pozyx.getDeviceCoordinates(device_ids[i], &anchor_coor);
    Serial.print(anchor_coor.x);
    print_comma();
    Serial.print(anchor_coor.y);
    print_comma();
    Serial.println(anchor_coor.z);
  }
}

// get ranges for each anchor
void get_ranges()
{
    // get range for each anchor
    device_range_t range;
    bool success = false;
    for (uint8_t i=0; i<NUM_ANCHORS; i++) {
        if (Pozyx.doRanging(anchor_id[i], &range) == POZYX_SUCCESS) {
            // send info to ardupilot
            send_beacon_distance(i, range.distance);
            success = true;
```

```
        }
    }

    // display errors
    if (!success) {
        Serial.println("failed to get any ranges");
    }
}

// get position of tag
void get_position()
{
    coordinates_t position;
    pos_error_t pos_error;

    //if (Pozyx.doPositioning(&position, POZYX_2_5D, 0) ==
POZYX_SUCCESS) {
    if (Pozyx.doPositioning(&position, POZYX_3D, 0, 0x00) ==
POZYX_SUCCESS) {
        if (Pozyx.getPositionError(&pos_error) == POZYX_SUCCESS) {
            // display position
            print_coordinates(position, pos_error);
            // send to ardupilot
            send_vehicle_position(position, pos_error);
        }
    } else {
        // display errors
        Serial.println("failed to calc position");
    }
}

// send all beacon config to ardupilot
void send_beacon_config()
{
    beacon_config_msg msg;
    msg.info.beacon_count = NUM_ANCHORS;
    for (uint8_t i=0; i<NUM_ANCHORS; i++) {
        msg.info.beacon_id = i;
        msg.info.x = anchors_x[i];
        msg.info.y = anchors_y[i];
        msg.info.z = heights[i];
        send_message(MSGID_BEACON_CONFIG, sizeof(msg.buf), msg.buf);
    }
    Serial.println("Sent anchor info");
}

// send a beacon's distance to ardupilot
void send_beacon_distance(uint8_t beacon_id, uint32_t distance_mm)
{
    beacon_distance_msg msg;
    msg.info.beacon_id = beacon_id;
```

```
    msg.info.distance = distance_mm;
    send_message(MSGID_BEACON_DIST, sizeof(msg.buf), msg.buf);
}

// send vehicle's position to ardupilot
void send_vehicle_position(coordinates_t& position, pos_error_t&
pos_error)
{
    vehicle_position_msg msg;

    // sanity check position
    if (position.x == 0 || position.y == 0) {
        return;
    }

    msg.info.x = position.x;
    msg.info.y = position.y;
    //msg.info.z = position.z;
    msg.info.z = 0;
    msg.info.position_error = pos_error.xy;
    send_message(MSGID_POSITION, sizeof(msg.buf), msg.buf);
}

void send_message(uint8_t msg_id, uint8_t data_len, uint8_t data_buf[])
{
    // sanity check
    if (data_len == 0) {
        return;
    }

    // message is buffer length + 1 (for checksum)
    uint8_t msg_len = data_len+1;

    // calculate checksum and place in last element of array
    uint8_t checksum = 0;
    checksum ^= msg_id;
    checksum ^= msg_len;
    for (uint8_t i=0; i<data_len; i++) {
        checksum = checksum ^ data_buf[i];
    }

    // send message
    int16_t num_sent = 0;
    num_sent += fcboardSerial.write(MSG_HEADER);
    num_sent += fcboardSerial.write(msg_id);
    num_sent += fcboardSerial.write(msg_len);
    num_sent += fcboardSerial.write(data_buf, data_len);
    num_sent += fcboardSerial.write(&checksum, 1);
    fcboardSerial.flush();
```

```
}
```