Direction Finding for 2.4 GHz

Direction finding is a technique within RF-communication that uses different applications for reception of radio waves to determine the direction of the beacon. By measuring the signals in an antenna array in phase and anti-phase, the sum-difference between the respective signals can be used to detect beacons by rotating the antenna in the azimuth- and elevation plane. The goal for the project is to design and construct an antenna array with the applicable characteristics, design and dimension the robot arm holding and orienting the receiver antenna and develop a program code for the search algorithm and data processing. The application utilizes the Bluetooth Low Energy as the network protocol and operates on 2.4 GHz. By researching the existing literature within the field of science, a functioning direction finding prototype is going to be developed. Through empirical testing and observation, the limits and constraints are going to be derived from the given system.

Generated by Doxygen 1.9.3

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 data_a Struct Reference

Encoder values for zero-points This struct stores the encoder values for the zero-points.

`#include <data_processor.h>`

Data Fields

- int16_t azimuth
- int16_t elevation

3.1.1 Detailed Description

Encoder values for zero-points This struct stores the encoder values for the zero-points.

3.1.2 Field Documentation

3.1.2.1 azimuth

`int16_t azimuth`

3.1.2.2 elevation

`int16_t elevation`

The documentation for this struct was generated from the following file:

- data_processor.h

3.2 data_s Struct Reference

Values measured at one point. This struct is used to store the measured values for encoder, RSSI at delta (phase-shifted) and RSSI for zigma(phase) at each measuring point.

`#include <data_processor.h>`

Data Fields

- int16_t encoder
- int16_t delta
- int16_t zigma

3.2.1 Detailed Description

Values measured at one point. This struct is used to store the measured values for encoder, RSSI at delta (phase-shifted) and RSSI for zigma(phase) at each measuring point.

3.2.2 Field Documentation

3.2.2.1 delta

```
int16_t delta
```

3.2.2.2 encoder

```
int16_t encoder
```

3.2.2.3 zigma

```
int16_t zigma
```

The documentation for this struct was generated from the following file:

- data_processor.h

Chapter 4

File Documentation

4.1 buttons.h File Reference

```
#include <dk_buttons_and_leds.h>
#include <drivers/gpio.h>
#include "search.h"
#include "laser.h"
```

Functions

- int configure_dk_buttons_leds ()

  Configures buttons and leds This function initiates the four leds and a button_handler which are connected to the buttons on the DK.

4.1.1 Function Documentation

4.1.1.1 configure_dk_buttons_leds()

```
int configure_dk_buttons_leds ( )
```

Configures buttons and leds This function initiates the four leds and a button_handler which are connected to the buttons on the DK.

Returns

int 0 when successful

4.2 buttons.h

Go to the documentation of this file.

```
1 #include <dk_buttons_and_leds.h>
2 #include <drivers/gpio.h>
3 #include "search.h"
4 #include "laser.h"
5
12 int configure_dk_buttons_leds();
```

4.3 data_processor.h File Reference

```
#include "encoder.h"
#include <zephyr/types.h>
```

Data Structures

- struct data_s

  Values measured at one point. This struct is used to store the measured values for encoder, RSSI at delta (phase-shifted) and RSSI for zigma(phase) at each measuring point.
- struct data_a

  Encoder values for zero-points This struct stores the encoder values for the zero-points.

Macros

- #define MIN_VALID_RSSI -90

  Minimum value for RSSI Minimum valid value for RSSI measurement, if values goes under this it becomes invalid and the measurement will not be used.
- #define MAX_VALID_RSSI -25

  Maximum value for RSSI Maximum valid value for RSSI measurement, if values goes over this it becomes invalid and the measurement will not be used.

Typedefs

- typedef struct data_s matrix_x3

  Values measured at one point. This struct is used to store the measured values for encoder, RSSI at delta (phase-shifted) and RSSI for zigma(phase) at each measuring point.
- typedef struct data_a zeros

  Encoder values for zero-points This struct stores the encoder values for the zero-points.

Functions

- void send_data (int16_t rssi, int index, int state)

  Sends measured RSSI signal into array This function is used by "Observer.c" to send measured RSSI value to either array for delta measurements or zigma measurements (data_delta and data_zigma). The array is later used by "get_average" calculate the average value for a more specific reading.
- void set_average_counter (int16_t value)

  Set the average counter object This function is used to set how many RSSI values should be measured before calculating the average RSSI value. If counter has a higher cap on 10 and lower cap on 1.
- int16_t get_average (int16_t list[ ])

  Get the average object This function is used to get the average RSSI value from array of RSSI values.
- void get_data (matrix_x3 ∗buffer_data, int N)

  Get the data object This function takes in a "matrix_x3" and transfers the current stored Delta, Zigma and encoder value into the variable.
- void value_validater (matrix_x3 ∗raw_data, int16_t ∗n)

  Validates the measured values This function checks the array consist of measured values are either over maximum set value "MAX_VALID_RSSI" or under "MIN_VALID RSSI". If the values are outside the allowed are the array at the index is set to 0.
- void update_matrix (matrix_x3 ∗data, int16_t ∗n)

  Updates array This function checks if some of the indexes of the array is set to 0. It will then delete that index and update the size of the variable n.
- bool zero_point_validater (int16_t value_zigma, int16_t value_delta, int16_t ZIGMA_ZERO_VALUE)

  Checks if the measured point is an valid null-point This function checks if the Delta value is smaller than the Zigma value and if the Zigma value is smaller than the "ZIGMA_ZERO_VALUE". Returns true if the conditions are met and false else.
- int find_zero_point (matrix_x3 validated_values[ ], int n)

  Finds the encoder zero point value of search This function starts with calling "find_zigma_zero_value". Next it will find the first valid zero-point and go through the rest of the array to find and return the lowest valid zero-point.
- int16_t find_zigma_zero_value (matrix_x3 values[ ], int n)

  Finds new ZIGMA_ZERO_VALUE This function takes in a array of measured values and finds the highest value for Zigma that are used to verify the zero-point. It will subtract the highest value by 3 and return this value.

4.3.1 Macro Definition Documentation

4.3.1.1 MAX_VALID_RSSI

```
#define MAX_VALID_RSSI -25
```

Maximum value for RSSI Maximum valid value for RSSI measurement, if values goes over this it becomes invalid and the measurement will not be used.

4.3.1.2 MIN_VALID_RSSI

```
#define MIN_VALID_RSSI -90
```

Minimum value for RSSI Minimum valid value for RSSI measurement, if values goes under this it becomes invalid and the measurement will not be used.

4.3.2 Typedef Documentation

4.3.2.1 matrix_x3

```
typedef struct data_s matrix_x3
```

Values measured at one point. This struct is used to store the measured values for encoder, RSSI at delta (phase-shifted) and RSSI for zigma(phase) at each measuring point.

4.3.2.2 zeros

```
typedef struct data_a zeros
```

Encoder values for zero-points This struct stores the encoder values for the zero-points.

4.3.3 Function Documentation

4.3.3.1 find_zero_point()

```
int find_zero_point (
            matrix_x3 validated_values[],
            int n )
```

Finds the encoder zero point value of search This function starts with calling "find_zigma_zero_value". Next it will find the first valid zero-point and go through the rest of the array to find and return the lowest valid zero-point.

Parameters

validated_values	Array of measured values
n	Size of array

Returns

index of zero-point

4.3.3.2 find_zigma_zero_value()

```
int16_t find_zigma_zero_value (
            matrix_x3 values[],
            int n )
```

Finds new ZIGMA_ZERO_VALUE This function takes in a array of measured values and finds the highest value for Zigma that are used to verify the zero-point. It will subtract the highest value by 3 and return this value.

Parameters

values	Array of measured values
n	Size of array

Returns

int16_t Modified ZIGMA_ZERO_VALUE

4.3.3.3 get_average()

```
int16_t get_average (
            int16_t list[] )
```

Get the average object This function is used to get the average RSSI value from array of RSSI values.

Parameters

list	List of RSSI values

Returns

Average value from list input

4.3.3.4 get_data()

```
void get_data (
            matrix_x3 * buffer_data,
            int N )
```

Get the data object This function takes in a "matrix_x3" and transfers the current stored Delta, Zigma and encoder value into the variable.

Parameters

buffer_data	A pointer to a "matrix_x3" variable
N	0 for Azimuth and 1 for Elevation

4.3.3.5 send_data()

```
void send_data (
            int16_t rssi,
            int index,
            int state )
```

Sends measured RSSI signal into array This function is used by "Observer.c" to send measured RSSI value to either array for delta measurements or zigma measurements (data_delta and data_zigma). The array is later used by "get_average" calculate the average value for a more specific reading.

Parameters

rssi	Value thats gone through Kalman filter
index	Index for where to store in the array
state	0 for delta and 1 for zigma

4.3.3.6 set_average_counter()

```
void set_average_counter (
            int16_t value )
```

Set the average counter object This function is used to set how many RSSI values should be measured before calculating the average RSSI value. If counter has a higher cap on 10 and lower cap on 1.

Parameters

value	Counter value

4.3.3.7 update_matrix()

```
void update_matrix (
            matrix_x3 * data,
            int16_t * n )
```

Updates array This function checks if some of the indexes of the array is set to 0. It will then delete that index and update the size of the variable n.

Parameters

data	A pointer to array with measured values
n	A pointer to size of the array

4.3.3.8 value_validater()

```
void value_validater (
            matrix_x3 * raw_data,
            int16_t * n )
```

Validates the measured values This function checks the array consist of measured values are either over maximum set value "MAX_VALID_RSSI" or under "MIN_VALID RSSI". If the values are outside the allowed are the array at the index is set to 0.

Parameters

raw_data	A pointer to array with measured values
n	A pointer to size of the array

4.3.3.9 zero_point_validater()

```
bool zero_point_validater (
            int16_t value_zigma,
            int16_t value_delta,
            int16_t ZIGMA_ZERO_VALUE )
```

Checks if the measured point is an valid null-point This function checks if the Delta value is smaller than the Zigma value and if the Zigma value is smaller than the "ZIGMA_ZERO_VALUE". Returns true if the conditions are met and false else.

Parameters

value_zigma	Zigma RSSI value
value_delta	Delta RSSI value
ZIGMA_ZERO_VALUE	

Returns

true If the condition are met

false If the conditions ar enot met

4.4 data_processor.h

Go to the documentation of this file.

```
1 #include "encoder.h"
2 #include <zephyr/types.h>
3
9 #define MIN_VALID_RSSI -90
15 #define MAX_VALID_RSSI -25
22 typedef struct data_s
23 {
24     int16_t encoder;
25     int16_t delta;
26     int16_t zigma;
27 } matrix_x3;
28
```

```
34 typedef struct data_a
35 {
36     int16_t azimuth;
37     int16_t elevation;
38 } zeros;
39
50 void send_data(int16_t rssi, int index, int state);
51
58 void set_average_counter(int16_t value);
59
66 int16_t get_average(int16_t list[]);
67
75 void get_data(matrix_x3 *buffer_data, int N);
76
85 void value_validater(matrix_x3 *raw_data, int16_t *n);
86
95 void update_matrix(matrix_x3 *data, int16_t *n);
96
109 bool zero_point_validater(int16_t value_zigma, int16_t value_delta, int16_t ZIGMA_ZERO_VALUE);
110
119 int find_zero_point(matrix_x3 validated_values[], int n);
120
129 int16_t find_zigma_zero_value(matrix_x3 values[], int n);
130
```

4.5 encoder.h File Reference

```
#include <logging/log.h>
#include "servo.h"
#include <nrfx_qdec.h>
```

Macros

- #define pin_a_azimuth 26

  Pin for Encoder azimuth "a".
- #define pin_b_azimuth 27

  Pin for Encoder azimuth "b".
- #define pin_a_elevation 30

  Pin for Encoder elevation "a".
- #define pin_b_elevation 31

  Pin for Encoder elevation "b".
- #define starting_angle_azimuth 0

  Starting angle for servomotor azimuth.
- #define starting_angle_elevation 0

  Starting angle for servomotor elevation.

Functions

- int init_encoder_servos ()

  Initiates servos and encoder This function initiates servomotor for azimuth plane, elevation plane and for the antenna before it moves the servomotors to starting positions. The function then resets the encoder values and initialize the irq handler for qdec.
- int init_encoder_azimuth ()

  Initialize azimuth encoder This function starts by disabling any current active encoders before initializing the encoder given the parameters set. If no error occurs the encoder is enabled .
- int init_encoder_elevation ()

Initialize elevation encoder This function starts by disabling any current active encoders before initializing the encoder given the parameters set. If no error occurs the encoder is enabled .

- void update_encoder (int N)

  Updates encoder value Depending on the parameter N the function will update the encoder value and convert it into degrees (8192 pulses/360 degrees) = 23.

- void angle_slow_move (int N, uint32_t angle)

  Moves servomotor gradually Depending on the parameter N the function will convert the input angle into the servomotors raw angle. Further the function will call upon either "increment_servo" og "decrement_servo" and update the encoder value with "update_encoder".

- int16_t get_encoder (int N)

  Get the encoder object Depending on the parameter N the function will return the encoder values in degrees.

4.5.1 Macro Definition Documentation

4.5.1.1 pin_a_azimuth

```
#define pin_a_azimuth 26
```

Pin for Encoder azimuth "a".

4.5.1.2 pin_a_elevation

```
#define pin_a_elevation 30
```

Pin for Encoder elevation "a".

4.5.1.3 pin_b_azimuth

```
#define pin_b_azimuth 27
```

Pin for Encoder azimuth "b".

4.5.1.4 pin_b_elevation

```
#define pin_b_elevation 31
```

Pin for Encoder elevation "b".

4.5.1.5 starting_angle_azimuth

```
#define starting_angle_azimuth 0
```

Starting angle for servomotor azimuth.

4.5.1.6 starting_angle_elevation

```
#define starting_angle_elevation 0
```

Starting angle for servomotor elevation.

4.5.2 Function Documentation

4.5.2.1 angle_slow_move()

```
void angle_slow_move (
            int N,
            uint32_t angle )
```

Moves servomotor gradually Depending on the parameter N the function will convert the input angle into the servomotors raw angle. Further the function will call upon either "increment_servo" og "decrement_servo" and update the encoder value with "update_encoder".

Parameters

N	0 for azimuth servomotor, 1 for elevation servomotor
angle	Working angles for the robot (0-180) azimuth, (0-70) elevation.

4.5.2.2 get_encoder()

```
int16_t get_encoder (
            int N )
```

Get the encoder object Depending on the parameter N the function will return the encoder values in degrees.

Parameters

N	0 for azimuth, 1 for elevation

Returns

int16_t Encoder value in degrees

4.5.2.3 init_encoder_azimuth()

```
int init_encoder_azimuth ( )
```

Initialize azimuth encoder This function starts by disabling any current active encoders before initializing the encoder given the parameters set. If no error occurs the encoder is enabled .

Returns

int NRFX_SUCCESS if successful

4.5.2.4 init_encoder_elevation()

```
int init_encoder_elevation ( )
```

Initialize elevation encoder This function starts by disabling any current active encoders before initializing the encoder given the parameters set. If no error occurs the encoder is enabled .

Returns

int NRFX_SUCCESS if successful

4.5.2.5 init_encoder_servos()

```
int init_encoder_servos ( )
```

Initiates servos and encoder This function initiates servomotor for azimuth plane, elevation plane and for the antenna before it moves the servomotors to starting positions. The function then resets the encoder values and initialize the irq handler for qdec.

Returns

int 0 if successful

4.5.2.6 update_encoder()

```
void update_encoder (
            int N )
```

Updates encoder value Depending on the parameter N the function will update the encoder value and convert it into degrees (8192 pulses/360 degrees) = 23.

Parameters

N	0 for azimuth and 1 elevation

4.6 encoder.h

Go to the documentation of this file.

```
1 #include <logging/log.h>
2 #include "servo.h"
3 #include <nrfx_qdec.h>
4
5
9 #define pin_a_azimuth 26
13 #define pin_b_azimuth 27
17 #define pin_a_elevation 30
21 #define pin_b_elevation 31
25 #define starting_angle_azimuth 0
29 #define starting_angle_elevation 0
30
39 int init_encoder_servos();
40
47 int init_encoder_azimuth();
48
55 int init_encoder_elevation();
56
63 void update_encoder(int N);
64
74 void angle_slow_move(int N, uint32_t angle);
75
82 int16_t get_encoder(int N);
83
```

4.7 initiater.h File Reference

```
#include "buttons.h"
```

Functions

- int initiate_modules ()

  Initiate necessary modules This function initiates buttons, leds, timer, BLE, servomotors, laser and azimuth encoder. Sets the average_counter to 1 and turns of the laser. If an error occurs while initiating a module it will print and return the error code and exit the function.

4.7.1 Function Documentation

4.7.1.1 initiate_modules()

```
int initiate_modules ( )
```

Initiate necessary modules This function initiates buttons, leds, timer, BLE, servomotors, laser and azimuth encoder. Sets the average_counter to 1 and turns of the laser. If an error occurs while initiating a module it will print and return the error code and exit the function.

Returns

int 0 when successful

4.8 initiater.h

Go to the documentation of this file.

```
1 #include "buttons.h"
11 int initiate_modules();
```

4.9 laser.h File Reference

```
#include <zephyr.h>
#include <sys/printk.h>
#include <device.h>
#include <devicetree.h>
#include <drivers/gpio.h>
```

Functions

- int laser_init ()

  Initialize laser pointer.
- void laser_toggle (void)

  Toggles laser pointer on or off.
- void laser_set (int state)

  Set laser to state.

4.9.1 Function Documentation

4.9.1.1 laser_init()

```
int laser_init ( )
```

Initialize laser pointer.

Parameters

laser_pin	Pin connected to laser +5v (nRF52 not compatible with 5v on gpio).

4.9.1.2 laser_set()

```
void laser_set (
            int state )
```

Set laser to state.

Parameters

state	int, 0 for on, 1 for off.

4.9.1.3 laser_toggle()

```
void laser_toggle (
            void  )
```

Toggles laser pointer on or off.

4.10 laser.h

Go to the documentation of this file.

```
1 #include <zephyr.h>
2 #include <sys/printk.h>
3 #include <device.h>
4 #include <devicetree.h>
5 #include <drivers/gpio.h>
6
12 int laser_init();
13
18 void laser_toggle(void);
19
25 void laser_set(int state);
```

4.11 main.c File Reference

```
#include "initiater.h"
#include "nrfx_qdec.h"
#include <stdio.h>
```

Functions

- K_SEM_DEFINE (my_sem, 0, 1)
- void main (void)

Variables

- zeros zero_enc_values

4.11.1 Function Documentation

4.11.1.1 K_SEM_DEFINE()

```
K_SEM_DEFINE (
             my_sem ,
             0 ,
             1  )
```

4.11.1.2 main()

```
void main (
             void  )
```

4.11.2 Variable Documentation

4.11.2.1 zero_enc_values

```
zeros zero_enc_values
```

4.12 observer.h File Reference

```
#include <bluetooth/bluetooth.h>
#include "data_processor.h"
```

Functions

- int add_filter_accept_list_from_string (const char *addr_str, const char *type)

  Adds address to filter This function takes in an address and adress type in form of a string and converts this into an ble_address. The address is then added into the filter accept list.
- int init_bluethooth_scan ()

  Initiates bluethooth This function starts by defining the parameters for the observer handler, setting the device tree to "GPIO_0", further the function enables bluethooth, adds address to filter list, initiates and starts the Observer handler and initiate the GPIO used for the switch.
- void set_observer (bool state)

  Set the observer object This function is being used by "sweep_search" to activate the logic inside the observer handler.

4.12.1 Function Documentation

4.12.1.1 add_filter_accept_list_from_string()

```
int add_filter_accept_list_from_string (
             const char * addr_str,
             const char * type )
```

Adds address to filter This function takes in an address and adress type in form of a string and converts this into an ble_address. The address is then added into the filter accept list.

Parameters

addr_str	Address in form of a string
type	Type in form of a string ("random") or ("public")

Returns

int 0 if successful

4.12.1.2 init_bluethooth_scan()

```
int init_bluethooth_scan ( )
```

Initiates bluethooth This function starts by defining the parameters for the observer handler, setting the device tree to "GPIO_0", further the function enables bluethooth, adds address to filter list, initiates and starts the Observer handler and initiate the GPIO used for the switch.

Returns

int 0 if successful

4.12.1.3 set_observer()

```
void set_observer (
            bool state )
```

Set the observer object This function is being used by "sweep_search" to activate the logic inside the observer handler.

Parameters

state	false or true

4.13 observer.h

Go to the documentation of this file.

```
1 #include <bluetooth/bluetooth.h>
2 #include "data_processor.h"
3
4
5
17 int add_filter_accept_list_from_string(const char *addr_str,const char *type);
18
28 int init_bluethooth_scan();
29
44 static void device_found(const bt_addr_le_t *addr, int8_t rssi, uint8_t type, struct net_buf_simple *ad);
```

```
45
51 void set_observer(bool state);
52
53
54
55
56
57
58
59
60
61
```

4.14 search.h File Reference

```
#include "observer.h"
```

Macros

- #define MOTOR_TEST 0
- #define SEARCH_AZIMUTH 1

  Activates search in azimuth 0 to deactivate search in azimuth, 1 to activate search in azimuth.
- #define SEARCH_ELEVATION 1
- #define FINE_ACTIVATE 1

  Activates fine search 0 to deactivate fine search, 1 to activate fine search.

Functions

- void sweep_search (int state, int16_t min_encoder_search, int16_t max_encoder_search, int increment)

  Search and stores RSSI values within given area This function changes the value for servo angle variable that the Threads use to move the servo. Further the function activates the logic in the observer handler and waits until the semaphore is released from the handler before it gets and transfer the data to readings array. It does this for the whole sweep sector before releasing the semaphore.
- int get_readings (matrix_x3 *main_readings, int16_t *n)

  Get the readings object This function is used to validate, update and retrieve the readings that were done by sweep↩ _search before resetting the array "readings".
- int16_t fine_sweeper (int state, int threshold_degrees, int threshold_search, int sweep_sector, int16_t zero↩ _point)

  Fine search of RSSI values This function starts by calculating the new min_encoder_value and max_encoder_value based on the zero_point and the sweep_sector. Further the function sets the average counter to 10 to get more accurate readings. Then it calls on "sweep_search" with the new calculated parameters, gets the measurements and finds the zero-point index. The readings are then analysed for finding a new sweeps ector based on the delta value at the zero-point index and threashold_degrees. Depending if the new sweep sector is under threshold_search the function either exits or calls upon itself (recursion) with the new parameters calculated.
- void reset_readings ()

  Sets values to 0 This function set all the values of the array "readings" to 0.
- zeros coarse_search ()

  Coarse search.
- zeros fine_search (zeros enc_values)

  Fine search This function starts by calling upon "fine_sweeper" with the parameters given and finds the encoder values for the zero-point before it validates that the servo is at zero-point. If SEARCH_PLANE is defined to be higher than 1 it starts by turning the antenna 90 degrees and searches in the elevation plane. When switching search plane the thread and encoder that is not being searched are disabled.
- void validate_servo_zero_moved (int N, uint32_t zero_point_servo_angle)

Validates servomotor position This function checks if the servomotor is at angle given from parameters. It is used to wait for the motor is at correct position before the code starts searching in another plane.

- void azimuth_servo_thread (uint32_t *azimuth_thread_servo_angle)

  Moves azimuth servo Thread that is used to move servomotor for azimuth plane.
- void elevation_servo_thread (uint32_t *elevation_thread_servo_angle)

  Moves elevation servo Thread that is used to move servomotor for elevation plane.

4.14.1 Macro Definition Documentation

4.14.1.1 FINE_ACTIVATE

```
#define FINE_ACTIVATE 1
```

Activates fine search 0 to deactivate fine search, 1 to activate fine search.

4.14.1.2 MOTOR_TEST

```
#define MOTOR_TEST 0
```

4.14.1.3 SEARCH_AZIMUTH

```
#define SEARCH_AZIMUTH 1
```

Activates search in azimuth 0 to deactivate search in azimuth, 1 to activate search in azimuth.

4.14.1.4 SEARCH_ELEVATION

```
#define SEARCH_ELEVATION 1
```

4.14.2 Function Documentation

4.14.2.1 azimuth_servo_thread()

```
void azimuth_servo_thread (
            uint32_t * azimuth_thread_servo_angle )
```

Moves azimuth servo Thread that is used to move servomotor for azimuth plane.

Parameters

azimuth_thread_servo_angle	

4.14.2.2 coarse_search()

```
zeros coarse_search ( )
```

Coarse search.

- This function starts search in azimuth plane by calling upon the "sweep_search" and uses "find_zero_point" to find the encoder value for the zero-point. After finding the zero-point it uses "validate_servo_zero_moved" to validate that the servomotor is at zero-point. If SEARCH_PLANE is defined to be higher than 1 it starts by turning the antenna 90 degrees and searches in the elevation plane. When switching search plane the thread and encoder that is not being searched are disabled.

Returns

zeros Encoder values for zero-point in azimuth and elevation

4.14.2.3 elevation_servo_thread()

```
void elevation_servo_thread (
            uint32_t * elevation_thread_servo_angle )
```

Moves elevation servo Thread that is used to move servomotor for elevation plane.

Parameters

elevation_thread_servo_angle	

4.14.2.4 fine_search()

```
zeros fine_search (
            zeros enc_values )
```

Fine search This function starts by calling upon "fine_sweeper" with the parameters given and finds the encoder values for the zero-point before it validates that the servo is at zero-point. If SEARCH_PLANE is defined to be higher than 1 it starts by turning the antenna 90 degrees and searches in the elevation plane. When switching search plane the thread and encoder that is not being searched are disabled.

Parameters

enc_values	Zero-point values for azimuth and elevation plane

Returns

zeros Encoder values for zero-point in azimuth and elevation

4.14.2.5 fine_sweeper()

```
int16_t fine_sweeper (
            int state,
            int threshold_degrees,
            int threshold_search,
            int sweep_sector,
            int16_t zero_point )
```

Fine search of RSSI values This function starts by calculating the new min_encoder_value and max_encoder_value based on the zero_point and the sweep_sector. Further the function sets the average counter to 10 to get more accurate readings. Then it calls on "sweep_search" with the new calculated parameters, gets the measurements and finds the zero-point index. The readings are then analysed for finding a new sweeps ector based on the delta value at the zero-point index and threashold_degrees. Depending if the new sweep sector is under threshold_search the function either exits or calls upon itself (recursion) with the new parameters calculated.

Parameters

state	0 for Delta 1 for Zigma
threshold_degrees	Threshold for RSSI value when defining new sweep sector
threshold_search	Threshold for exiting function
sweep_sector	Value for sweep sector
zero_point	Encoder value for zero-point

Returns

int16_t Encoder value for zero-point

4.14.2.6 get_readings()

```
int get_readings (
            matrix_x3 * main_readings,
            int16_t * n )
```

Get the readings object This function is used to validate, update and retrieve the readings that were done by sweep_search before resetting the array "readings".

Parameters

main_readings	Pointer to where the array where values are being stored
n	Pointer to size of array

Returns

int 0 when successfull

4.14.2.7 reset_readings()

```
void reset_readings ( )
```

Sets values to 0 This function set all the values of the array "readings" to 0.

Returns

zeros 0 when successful

4.14.2.8 sweep_search()

```
void sweep_search (
            int state,
            int16_t min_encoder_search,
            int16_t max_encoder_search,
            int increment )
```

Search and stores RSSI values within given area This function changes the value for servo angle variable that the Threads use to move the servo. Further the function activates the logic in the observer handler and waits until the semaphore is released from the handler before it gets and transfer the data to readings array. It does this for the whole sweep sector before releasing the semaphore.

Parameters

state	0 for azimuth, 1 for Elevation
min_encoder_search	Minimum search degrees for search sector
max_encoder_search	Maximum search degrees for search sector
increment	Value for increment for degrees

4.14.2.9 validate_servo_zero_moved()

```
void validate_servo_zero_moved (
```

```
            int N,
            uint32_t zero_point_servo_angle )
```

Validates servomotor position This function checks if the servomotor is at angle given from parameters. It is used to wait for the motor is at correct position before the code starts searching in another plane.

Parameters

N	0 for azimuth servomotor, 1 for elevation servomotor
zero_point_servo_angle	Value for servomotor angle

4.15 search.h

Go to the documentation of this file.

```
1 #include "observer.h"
2
3 #define MOTOR_TEST 0
4
10 #define SEARCH_AZIMUTH 1
11 /*0
12  * @brief Activates serach in elevation
13  * 0 to deactivate search in elevation, 1 to activate search in elevation
14  */
15 #define SEARCH_ELEVATION 1
20 #define FINE_ACTIVATE 1
21
22
36 void sweep_search(int state, int16_t min_encoder_search, int16_t max_encoder_search, int increment);
37
46 int get_readings(matrix_x3 *main_readings, int16_t *n);
47
68 int16_t fine_sweeper(int state, int threshold_degrees, int threshold_search, int sweep_sector, int16_t
       zero_point);
69
75 void reset_readings();
76
77
90 zeros coarse_search();
91
102 zeros fine_search(zeros enc_values);
103
113 void validate_servo_zero_moved(int N, uint32_t zero_point_servo_angle);
114
121 void azimuth_servo_thread(uint32_t *azimuth_thread_servo_angle);
122
129 void elevation_servo_thread(uint32_t *elevation_thread_servo_angle);
```

4.16 servo.h File Reference

```
#include <zephyr.h>
#include <logging/log.h>
#include <drivers/gpio.h>
```

Macros

- #define servo_azimuth_N 0

  N for azimuth servomotor.
- #define servo_azimuth_pin 2

  Pin for azimuth servomotor.

- #define servo_elevationl_N 1

  N for elevation servomotor.
- #define servo_elevation_pin 3

  Pin for elevation servomotor.
- #define servo_antenna_N 2

  N for antenna servomotor.
- #define servo_antenna_pin 4

  Pin for antenne servomotor.

Functions

- int timer_init ()

  Init timer.
- int timer_start ()

  Start timer.
- int servo_init (uint32_t N, int servo_pin)

  Funciton that initializes servo on specified pin. IMPORTANT, for servos to behave correctly, all prior servos N have to be initialized, and in order. That is, if initializing servo N = 1, servo N = 0 has to also be initialized first.
- void angle_move_servo (int N, uint32_t angle)

  Move servo to angle given in degrees.
- void raw_move_servo (int N, uint32_t position)

  Move servo to position in ticks.
- uint32_t sin_scaled (uint32_t input, uint32_t input_max, uint32_t output_min, uint32_t output_max)
- void increment_servo (int N)

  Increments servo angle Depending on the parameter N the function will update the servomotor angle and use the function "raw_move_servo" and "convert_to_raw" to increment the servomotor angle.
- void decrement_servo (int N)

  Depending on the parameter N the function will update the servomotor angle and use the function "raw_move_servo" and "convert_to_raw" to decrement the servomotor angle.
- int16_t get_servo_angle (int N)

  Get the servo angle object Depending on the parameter N the function will return either the servomotor angle for either azimuth or elevation.

4.16.1 Macro Definition Documentation

4.16.1.1 servo_antenna_N

```
#define servo_antenna_N 2
```

N for antenna servomotor.

4.16.1.2 servo_antenna_pin

```
#define servo_antenna_pin 4
```

Pin for antenne servomotor.

4.16.1.3 servo_azimuth_N

```
#define servo_azimuth_N 0
```

N for azimuth servomotor.

4.16.1.4 servo_azimuth_pin

```
#define servo_azimuth_pin 2
```

Pin for azimuth servomotor.

4.16.1.5 servo_elevation_pin

```
#define servo_elevation_pin 3
```

Pin for elevation servomotor.

4.16.1.6 servo_elevationl_N

```
#define servo_elevationl_N 1
```

N for elevation servomotor.

4.16.2 Function Documentation

4.16.2.1 angle_move_servo()

```
void angle_move_servo (
            int N,
            uint32_t angle )
```

Move servo to angle given in degrees.

Parameters

N	Servo to be moved
angle	Angle given in degrees

4.16.2.2 decrement_servo()

```
void decrement_servo (
            int N )
```

Depending on the parameter N the function will update the servomotor angle and use the function "raw_move_↩ servo" and "convert_to_raw" to decrement the servomotor angle.

Parameters

N	0 for azimuth, 1 for elevation

4.16.2.3 get_servo_angle()

```
int16_t get_servo_angle (
            int N )
```

Get the servo angle object Depending on the parameter N the function will return either the servomotor angle for either azimuth or elevation.

Parameters

N	0 for azimuth, 1 for elevation

Returns

int16_t Servomotor angle

4.16.2.4 increment_servo()

```
void increment_servo (
            int N )
```

Increments servo angle Depending on the parameter N the function will update the servomotor angle and use the function "raw_move_servo" and "convert_to_raw" to increment the servomotor angle.

Parameters

N	0 for azimuth, 1 for elevation

4.16.2.5 raw_move_servo()

```
void raw_move_servo (
            int N,
            uint32_t position )
```

Move servo to position in ticks.

Raw position is given in ticks, in this case the timer is running at a frequency of 16MHz. We have a counter modulus, TIMER_RELOAD of 320000. This was found by dividing clock speed with servo frequency. If our servo needs input frequency of 50Hz, we divide 50^-1 by 320000 and get the period counter. If we know the dudy cycle of the servo, we can calculate the input needed to position the servo with: W = duty_cycle/period_counter For example, with neutral position in 1.5ms, we have: W = 0.0015/6.25∗10^-8 = 24000

Specs for MG90S, 0.5ms to 2.5ms, neutral at 1.5, 0deg to 180deg Specs for SER0038, 0.5ms = 0deg, 2.5ms = 270deg, 1.5 neutral

Parameters

N	Servo to be moved
position	Position in ticks

4.16.2.6 servo_init()

```
int servo_init (
            uint32_t N,
            int servo_pin )
```

Funciton that initializes servo on specified pin. IMPORTANT, for servos to behave correctly, all prior servos N have to be initialized, and in order. That is, if initializing servo N = 1, servo N = 0 has to also be initialized first.

Takes in two parameters

Parameters

N	Servo numer, from 0 to 3
servo_pin	Pin where servo is connected

4.16.2.7 sin_scaled()

```
uint32_t sin_scaled (
            uint32_t input,
            uint32_t input_max,
            uint32_t output_min,
            uint32_t output_max )
```

4.16.2.8 timer_init()

```
int timer_init ( )
```

Init timer.

4.16.2.9 timer_start()

```
int timer_start ( )
```

Start timer.

4.17 servo.h

Go to the documentation of this file.

```
1 #include <zephyr.h>
2 #include <logging/log.h>
3 #include <drivers/gpio.h>
4
8 #define servo_azimuth_N 0
12 #define servo_azimuth_pin 2
16 #define servo_elevationl_N 1
20 #define servo_elevation_pin 3
24 #define servo_antenna_N 2
28 #define servo_antenna_pin 4
29
34 int timer_init();
35
40 int timer_start();
41
51 int servo_init(uint32_t N, int servo_pin);
52
59 void angle_move_servo(int N, uint32_t angle);
60
80 void raw_move_servo(int N, uint32_t position);
81
82
83 uint32_t sin_scaled(uint32_t input, uint32_t input_max, uint32_t output_min, uint32_t output_max);
84
91 void increment_servo(int N);
92
99 void decrement_servo(int N);
100
108 int16_t get_servo_angle(int N);
109
110
111
```

Index