POLITECNICO DI BARI

DIPARTIMENTO DI INGEGNERIA ELETTRICA E DELL'INFORMAZIONE
LAUREA MAGISTRALE IN INGEGNERIA DELL'AUTOMAZIONE

Mobile Robotics

Prof. Luca De Cicco

Title:

Object tracking of a differential drive robot

Student:
Damiano Vernice

ACADEMIC YEAR 2024-2025

Contents

1 Introduction

The objective of this project is to design and implement a differential-drive mobile robot capable of object tracking in space. The present report documents the hardware choices, electronic connections and power supply considerations, the software architecture and the control strategy employed to track and follow a target using a Time-of-Flight (ToF) sensor (VL53L5CX) interfaced with an STM32 NUCLEO board.

2 Hardware structure

A differential-drive robot chassis was purchased. It already includes two geared DC motors (gear ratio 1:120), two controlled wheels, and an universal wheel for balancing. An additional mechanical support was installed to hold the sensor in front of the robot.

From the electrical standpoint, the components used are:

- STM32-NUCLEO F446RE development board
- PALO battery, nominal 12.6 V, maximum output 5 A
- X-NUCLEO-IHM04A1 motor driver expansion board
- VL53L5CX-SATEL breakout board (Time-of-Flight multi-zone sensor)
- Bipolar on/off switch
- Soldered step-down (buck) voltage regulator

The bipolar switch was included to enable convenient power on/off of the vehicle. The battery provides power simultaneously to the motor driver and the microcontroller. The step-down regulator reduces the battery voltage to appropriate levels required by the microcontroller and peripheral devices.

2.1 VL53L5CX

For this application the VL53L5CX-SATEL breakout board was used instead of the bare module. The breakout board integrates pull-up resistors and provides a simpler mechanical/electrical interface with the microcontroller; additionally, it comes with a richer API. Despite this, the project purposely relied on the base library functions to increase awareness of the low-level sensor operations.

Technical summary of the VL53L5CX (STMicroelectronics):

- Multi-zone Time-of-Flight (ToF) sensor providing absolute distance measurements largely independent of target color and reflectance.
- Package: compact LGA16 optical package; integrates a SPAD array, diffractive optics (DOE), and IR filters, optimized for variable ambient conditions.
- Light source: VCSEL at 940 nm.
- Field of view: square 65° diagonal (approximately 45° horizontal and vertical).
- Measurement modes: continuous and autonomous; up to about 60 Hz depending on resolution and settings.
- Resolutions: 4×4 (16 zones) and 8×8 (64 zones) — the project used 8×8 to maximize spatial information density.

- Maximum range: up to 400 cm (configuration-dependent); typical accuracy: ±4% on high-reflectance targets, ±5% on dark targets (dependant on illumination and scene).
- Interface: I^2C (compatible up to 1 MHz), 7-bit device address often used in library as 0x29 (shifted to 0x52 for 8-bit addressing conventions). Registers use 16-bit addresses and data are typically transmitted as 7-bit bytes per I2C conventions.
- Power and pins: AVDD and IOVDD supply options, LPn (I^2C enable), PWREN (power enable), I2CRST (I2C reset), INT (interrupt), SCL and SDA.
- Low-power modes and on-device processing (the module has an internal MCU and memory to handle histogram processing and multi-target detection).

Because the VL53L5CX integrates histogram processing and multi-target logic, it is particularly suitable for robotics tasks such as obstacle detection, 3D mapping, gesture recognition and other applications where multi-zone distance data is advantageous.

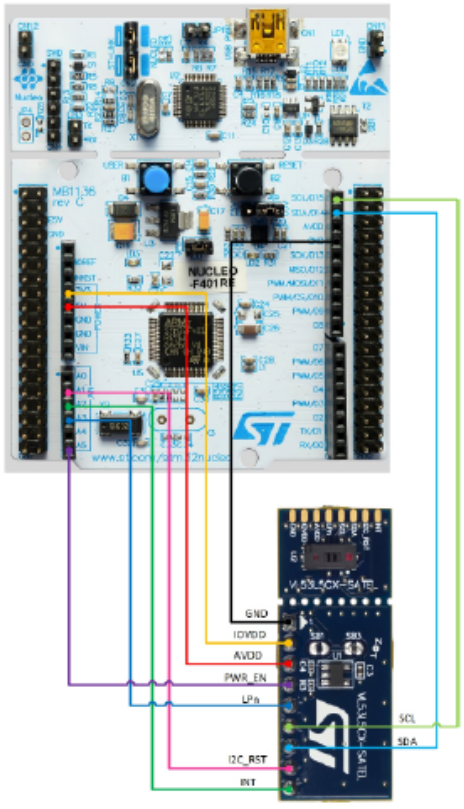

Figure 1: vl53l5cx wiring

2.2 X-NUCLEO-IHM04A1

The X-NUCLEO-IHM04A1 expansion board is a driver for brushed DC motors designed to stack with STM32 Nucleo boards. Its salient characteristics:

- Based on the L6206 DMOS dual full-bridge driver.
- Supports driving two bipolar DC motors or four unipolar DC motors (configuration via jumpers).
- Supply voltage: 8 V to 50 V.
- Phase current: up to 2.8 Arms per bridge, with parallel configurations enabling up to 11.2 Arms in total.
- Typical $R_{DS(on)}$: 0.3 Ω per MOSFET (reduced when bridges are paralleled).
- Overcurrent detection (OCD) with programmable thresholds (via external resistors).
- Multiple operational modes: independent bidirectional control, unidirectional configurations, parallel bridges for higher currents, etc.
- Arduino UNO R3 compatible connector and ST Morpho headers for stacking and expansion.
- On-board diodes for flyback protection, status LEDs and thermal dissipation considerations.
- Provided software package X-CUBE-SPN4 with motor control examples (not used in this project).

Overall, the X-NUCLEO-IHM04A1 offers a robust and modular solution to drive DC motors in embedded robotics applications with STM32 hardware.

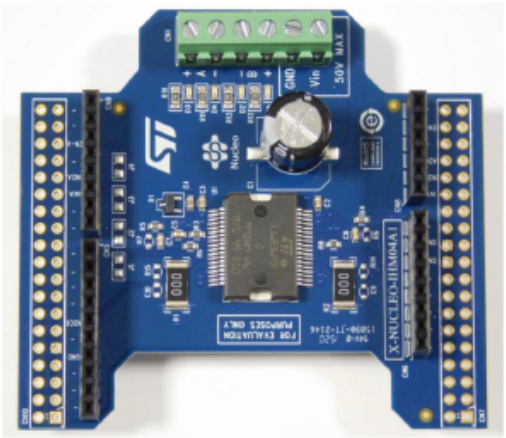

Figure 2: X-NUCLEO-IHM04A1

3 Software and Code

The STM32 NUCLEO F446RE microcontroller was programmed using STM32CubeIDE with C as the implementation language. The project exploits automatically generated initialization code for peripherals (timers, GPIO, I^2C), while custom application code implements sensor configuration, data acquisition and closed-loop feedback control.

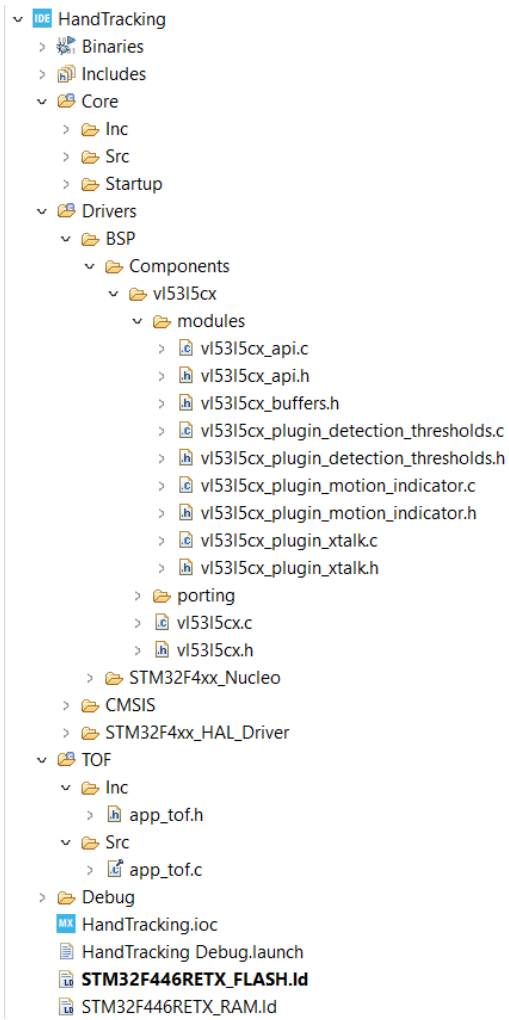

Figure 3: Files tree

The project tree (project explorer) contains both autogenerated files by STM32CubeIDE and several custom files. Specifically, the folder `vl53l5cx` contains the driver sources (".c" and ".h") used to interface and configure the VL53L5CX sensor, including optional modules such as:

- `vl53l5cx_plugin_detection_threshold` — threshold-based detection configuration
- `vl53l5cx_motion_indicator` — plugins to assess motion indicators on zones
- `vl53l5cx_plugin_xtalk` — cross-talk mitigation plugin

For the purposes of this experiment, only the core modules required for distance measurement were used; the optional plugins were not employed.

3.1 Pin Configuration

The main I/O configuration includes:

- Six dedicated pins connected to the VL53L5CX breakout (including INT, I2C reset, LPn and power enable), plus power lines (GND, IOVDD 5V, AVDD 3.3V)
- Six motor control pins to drive the two full-bridges of the motor driver (e.g. ENA, ENB and the timer PWM channels for the H-bridge inputs)
- I^2C SCL and SDA pins used for communication with the VL53L5CX

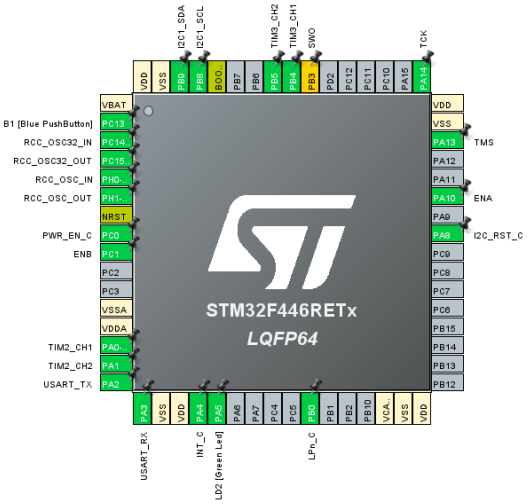

Figure 4: Pin configuration

3.2 Code

Below are the important parts from `main.c` and the sensor application files (`app_tof.c / .h`).

3.2.1 Starting PWM timers and sensor initialization

After the typical configuration autogenereted from the software, also the PWM timers for motor control and the TOF sensor are inizializated and configurated.

```
1 if (HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1) != HAL_OK) {
2     HardFault_Handler();
3 }
4
5 if (HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2) != HAL_OK) {
6     HardFault_Handler();
```

```
}

if (HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1) != HAL_OK) {
    HardFault_Handler();
}

if (HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2) != HAL_OK) {
    HardFault_Handler();
}

Tof_init();
Tof_conf();
```

Listing 1: Start PWM channels and start sensor setup

3.2.2 ToF initialization

This function configures the device structure for the VL53L5CX library, sets threshold defaults, resets the hardware and performs the library initialization. The code sets up the device callback functions for read/write/tick operations and configures a basic detection threshold window. The hardware reset ensures the sensor starts in a known state.

```
void Tof_init(void) {

    p_dev.platform.address = 0x29 << 1;
    p_dev.platform.Write = VL53L5CX_I2C_Write;
    p_dev.platform.Read = VL53L5CX_I2C_Read;
    p_dev.platform.GetTick = VL53L5CX_GetTick;

    p_thresholds.param_low_thresh = desired_distance + 25;
    p_thresholds.param_high_thresh = desired_distance - 25;
    p_thresholds.measurement = VL53L5CX_DISTANCE_MM;
    p_thresholds.type = VL53L5CX_OUT_OF_WINDOW;
    p_thresholds.zone_num = VL53L5CX_LAST_THRESHOLD;

    HardwareReset();

    printf("Starting VL53L5CX init ...\n");
    if (vl53l5cx_init(&p_dev) != VL53L5CX_OK) {
        printf("Init Error\n");
        Error_Handler();
    } else {
        printf("Init OK\n");
    }
}
```

Listing 2: Tof_init function

3.2.3 ToF configuration

The function sets the sensor to 8×8 resolution (chosen for higher fidelity), chooses the closest-target per zone policy, sets a sampling frequency (15 Hz was chosen as the maximum for 8×8), and starts continuous ranging. Thresholds on cells interrupt are disabled because of their useless since the sensor cover a huge visual in the space.

```
void Tof_conf(void) {

    if (vl53l5cx_set_resolution(&p_dev,
       VL53L5CX_RESOLUTION_8X8) != VL53L5CX_OK) {
        printf("Error: Set Resolution\n");
        Error_Handler();
    } else {
        printf("Set Resolution OK\n");
    }

    if (vl53l5cx_set_target_order(&p_dev,
       VL53L5CX_TARGET_ORDER_CLOSEST) != VL53L5CX_OK) {
        printf("Error: Set Target Order\n");
        Error_Handler();
    } else {
        printf("Set Target Order OK\n");
    }

    if (vl53l5cx_set_ranging_frequency_hz(&p_dev, frequency)
       != VL53L5CX_OK) {
        printf("Error: Set Ranging Frequency\n");
        Error_Handler();
    } else {
        printf("Set Ranging Frequency OK\n");
    }

    if (vl53l5cx_set_ranging_mode(&p_dev,
       VL53L5CX_RANGING_MODE_CONTINUOUS) != VL53L5CX_OK) {
        printf("Error: Set Ranging Mode\n");
        Error_Handler();
    } else {
        printf("Set Ranging Mode OK\n");
    }

    if (vl53l5cx_set_sharpener_percent(&p_dev,
       sharpener_percent) != VL53L5CX_OK) {
        printf("Error: Set Sharpener\n");
        Error_Handler();
    } else {
        printf("Set Sharpener OK\n");
    }

    if (vl53l5cx_set_detection_thresholds_enable(&p_dev, 0)
```

```
        != VL53L5CX_OK) {
39          printf("Error: Set Detection Thresholds Enable\n");
40          Error_Handler();
41      } else {
42          printf("Set Detection Thresholds Enable OK\n");
43      }
44
45
46      if (vl53l5cx_start_ranging(&p_dev) != VL53L5CX_OK) {
47          printf("Error: start ranging\n");
48          Error_Handler();
49      } else {
50          printf("Start OK\n");
51      }
52
53      ready = 1;
54  }
```

Listing 3: Tof_conf function

3.2.4 Interrupt callback

The VL53L5CX breakout asserts an interrupt line when new ranging data meet the detection criteria. The IRQ handler simply sets a flag so the main loop can process the data outside the interrupt context.

```
1  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
2      if (GPIO_Pin == INT_C_Pin) {
3          tof_data_ready = 1;
4      }
5  }
```

Listing 4: External interrupt callback

3.2.5 Main loop: data readiness and acquisition

In the microcontroller main loop the code checks two flags: `ready` (sensor initialized and configuration complete) and `tof_data_ready` (interrupt set). If both conditions are met, it queries the sensor library for data readiness and then reads ranging data. If the read is successful the feedback controller is called. The design choice to only set a minimal flag in the IRQ and defer the heavier processing to the main loop prevents nested or concurrent execution of the feedback routine.

```
1  while (1) {
2      if (tof_data_ready && ready == 1) {
3          tof_data_ready = 0;
4
5          uint8_t isReady;
6          if (vl53l5cx_check_data_ready(&p_dev, &isReady) ==
               VL53L5CX_STATUS_OK && isReady) {
```

```
            if (vl53l5cx_get_ranging_data(&p_dev, &results)
                != VL53L5CX_STATUS_OK) {
                printf("Error: GetRanging\n");
                HardFault_Handler();
            } else {
                feedbackcontroll();
            }
        }
    }
}
```

Listing 5: Main loop

3.3 Control Strategy

Control strategy was developed referring to lecture slides by Prof. Luca De Cicco (Mobile Robotics).

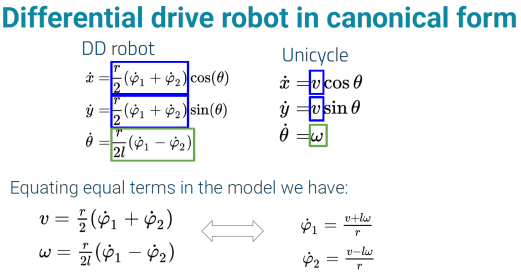

Figure 5: DDR in canonical form

From the kinematic model of a differential-drive robot, the canonical relationships between linear/angular velocities and wheel angular velocities are used.

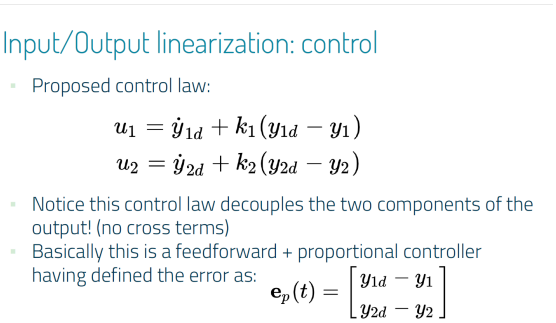

Figure 6: Control strategy

A simple proportional control (P controller) is implemented that maps errors in measured distance and lateral displacement (column index of the 8×8 sensor matrix) to linear and angular velocity commands. In the application, the desired linear and angular velocities $\dot{y}_1^d$ and $\dot{y}_2^d$ are set to zero (we want the vehicle to stop when it

reaches the desired position). The computed linear velocity and angular velocity are then converted to wheel angular velocities using the canonical kinematic equations, and finally into PWM duty cycles for motor drive.

3.4 Feedback control function

This section reproduces and explains the feedback control function that processes the 8×8 distance matrix from the VL53L5CX, extracts columns with valid targets, computes mean distances per column, decides the column with the largest number of detections and computes the control action.

3.4.1 Control-related global variables

These variables are declared outside the function and used by the feedback-control routine. Gains were tuned experimentally. Desired distance is set equal to 300 mm (30 cm) and center index to 4.5 since the presence of 8 columns. Some mechanical parameters and other useful variables are used to make computations.

```
float linear_gain = 5.0f;
float angular_gain = 5.0f;
float desired_distance = 300.0f;
float measured_distance;
float center = 4.5f;
float l = 5.0f;
float r = 3.25f;
float error_linear_vel;
float error_angular_vel;
float dc1;
float dc2;
float w;
```

Listing 6: Control variables

3.4.2 Computation of maximum values and per-column accumulators

Before reading the matrix, the function computes maximum expected velocities used for normalization and prepares counters and sums for each of the 8 columns.

```
float v_max = fabsf((desired_distance - (float)4000.0));
float w_max = fabsf(((float)8.0 - center));
float phi_max = (v_max + l * w_max) / r;


int num_col_1 = 0;
float sum_distance_col_1 = 0;


int num_col_2 = 0;
float sum_distance_col_2 = 0;

```

```
13
14 int num_col_3 = 0;
15 float sum_distance_col_3 = 0;
16
17
18 int num_col_4 = 0;
19 float sum_distance_col_4 = 0;
20
21
22 int num_col_5 = 0;
23 float sum_distance_col_5 = 0;
24
25
26 int num_col_6 = 0;
27 float sum_distance_col_6 = 0;
28
29
30 int num_col_7 = 0;
31 float sum_distance_col_7 = 0;
32
33
34 int num_col_8 = 0;
35 float sum_distance_col_8 = 0;
```

Listing 7: Initialization inside feedback function

3.4.3 Scan the 8×8 matrix and accumulate per-column statistics

The sensor provides 64 cells. The algorithm scans all zones and, for cells whose `target_status == 5` (indicating a reliable detection), it adds the measured distance to the corresponding column accumulator and increments the column counter. The code only considers cells with `target_status == 5` to increase reliability.

```
1  for (i = 0; i < 64; i++) {
2      if (results.target_status[i] == 5) {
3
4          if (i==0 || i==8 || i==16 || i==24 || i==32 || i==40
               || i==48 || i==56) {
5              num_col_1 += 1;
6              sum_distance_col_1 += results.distance_mm[i];
7          }
8
9
10         if (i==1 || i==9 || i==17 || i==25 || i==33 || i==41
               || i==49 || i==57) {
11             num_col_2 += 1;
12             sum_distance_col_2 += results.distance_mm[i];
13         }
14
15
```

```
16          if (i==2 || i==10 || i==18 || i==26 || i==34 || i==42
                || i==50 || i==58) {
17              num_col_3 += 1;
18              sum_distance_col_3 += results.distance_mm[i];
19          }
20
21
22          if (i==3 || i==11 || i==19 || i==27 || i==35 || i==43
                || i==51 || i==59) {
23              num_col_4 += 1;
24              sum_distance_col_4 += results.distance_mm[i];
25          }
26
27
28          if (i==4 || i==12 || i==20 || i==28 || i==36 || i==44
                || i==52 || i==60) {
29              num_col_5 += 1;
30              sum_distance_col_5 += results.distance_mm[i];
31          }
32
33
34          if (i==5 || i==13 || i==21 || i==29 || i==37 || i==45
                || i==53 || i==61) {
35              num_col_6 += 1;
36              sum_distance_col_6 += results.distance_mm[i];
37          }
38
39
40          if (i==6 || i==14 || i==22 || i==30 || i==38 || i==46
                || i==54 || i==62) {
41              num_col_7 += 1;
42              sum_distance_col_7 += results.distance_mm[i];
43          }
44
45
46          if (i==7 || i==15 || i==23 || i==31 || i==39 || i==47
                || i==55 || i==63) {
47              num_col_8 += 1;
48              sum_distance_col_8 += results.distance_mm[i];
49          }
50      }
51  }
```

Listing 8: Per-cell loop and per-column accumulation

3.4.4 Validity check and selection of the dominant column

After accumulation, the total number of columns (cells with valid target) is computed. If the total number of detected cells is below a threshold (6), the measurement

is deemed unreliable and the function returns early.

```
int total_cols = num_col_1 + num_col_2 + num_col_3 +
    num_col_4 + num_col_5 + num_col_6 + num_col_7 + num_col_8;

if (total_cols <= 5) {
    return;
}
```

Listing 9: Early exit condition

Then an array of counts is created and the column index with the maximum number of detections is found.

```
int num_cols[8] = {
    num_col_1, num_col_2, num_col_3, num_col_4,
    num_col_5, num_col_6, num_col_7, num_col_8
};

int max_index = 0;
int max_value = num_cols[0];

for (int k = 1; k < 8; k++) {
    if (num_cols[k] >= max_value) {
        max_value = num_cols[k];
        max_index = k;
    }
}
```

Listing 10: Index with highest count

3.4.5 Compute mean distance for the chosen column

Depending on the selected column, compute the mean distance from the sum and count:

```
if (max_index == 0) {
    measured_distance = sum_distance_col_1 / num_col_1;
} else if (max_index == 1) {
    measured_distance = sum_distance_col_2 / num_col_2;
} else if (max_index == 2) {
    measured_distance = sum_distance_col_3 / num_col_3;
} else if (max_index == 3) {
    measured_distance = sum_distance_col_4 / num_col_4;
} else if (max_index == 4) {
    measured_distance = sum_distance_col_5 / num_col_5;
} else if (max_index == 5) {
    measured_distance = sum_distance_col_6 / num_col_6;
} else if (max_index == 6) {
    measured_distance = sum_distance_col_7 / num_col_7;
} else if (max_index == 7) {
    measured_distance = sum_distance_col_8 / num_col_8;
```

```
}
```

Listing 11: Mean distance computation for selected column

3.4.6 Distance-based conditions and control law computation

The control uses proportional gains on distance error and column displacement. The center index and column thresholds produce zero angular velocity when the target is in central columns.

```
error_linear_vel = (measured_distance - desired_distance) ;

float v = linear_gain * error_linear_vel;

max_index ++;
if (max_index == 4 || max_index == 5) {

    error_angular_vel = 0.0;
    w = 0.0;
} else{

    error_angular_vel = (max_index - center);
    w = (angular_gain * error_angular_vel);
}

}
```

Listing 12: Distance checks and P-control

3.4.7 Saturation, deadband and PWM generation

The duty cycles are saturated and regulated to obtain a good behave of the vehicle. The PWM compare registers (CCR) are computed by scaling the absolute duty cycle to the timer period.

```
if (dc1 > 1.0f) dc1 = 1.0f;
if (dc1 < -1.0f) dc1 = -1.0f;

if (dc2 > 1.0f) dc2 = 1.0f;
if (dc2 < -1.0f) dc2 = -1.0f;

if (dc1 > 0.02f && dc1 < 0.2f) dc1 = 0.2f;
if (dc2 > 0.02f && dc2 < 0.2f) dc2 = 0.2f;

if (dc1 < -0.02f && dc1 > -0.2f) dc1 = -0.2f;
if (dc2 < -0.02f && dc2 > -0.2f) dc2 = -0.2f;

if ((dc1 > -0.02f && dc1 < 0.02f) && (dc1 > -0.02f && dc1 <
   0.02f)) dc1 = 0.0f;
```

```
14 if ((dc2 > -0.02f && dc2 < 0.02f) && (dc2 > -0.02f && dc2 <
       0.02f)) dc2 = 0.0f;
15
16
17 uint32_t ccr1 = (uint32_t) (fabsf(dc1) * (float) (1 +
       htim2.Init.Period));
18 uint32_t ccr2 = (uint32_t) (fabsf(dc2) * (float) (1 +
       htim3.Init.Period));
```

Listing 13: Saturation, deadband and PWM conversion

The code then maps the sign of the duty cycles to the appropriate PWM channels for direction (H-bridge requires separate channels for forward/reverse). Using separate compare channels for direction is standard for H-bridge control where one channel drives the forward switch and the other drives the reverse switch.

```
1  if (dc1 > 0.0f) {
2      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, ccr1);
3      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, 0);
4  } else if (dc1 == 0.0f) {
5      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
6      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, 0);
7  } else {
8      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, 0);
9      __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_2, ccr1);
10 }
11
12 if (dc2 > 0.0f) {
13     __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, ccr2);
14     __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, 0);
15 } else if (dc2 == 0.0f) {
16     __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, 0);
17     __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, 0);
18 } else {
19     __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, 0);
20     __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_2, ccr2);
21 }
```

Listing 14: Set PWM compare registers based on sign

4 Results

In this section it will show some results obtained. To demostrate that a control system is working properly is necessary to shown how control variables evolve in time and how control actions and errors achieve the desired results.

4.1 Linear velocity error

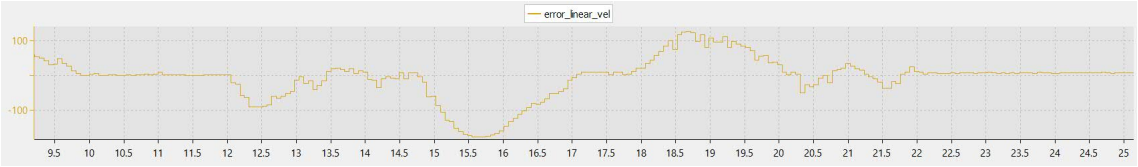

Figure 7: Linear velocity error

Here is shown the SWV Data Trace Timeline Graph in reference to the linear velocity error. Obviously, the peaks rappresents the moment when the target is moved and is clear that the veichle tends to compensate this error trying to chase it.

4.2 Angular velocity

Figure 8: Angular velocity

In this subsection, instead, is shown how the angular velocity evolves in time. Also in this case, is clear that the veichle tends to compensate its orientation error after a peak. It needs precisize that the values between -7.5 and 7.5 can be considered irrelevant in terms of control actions and just in few cases can took to an oscillant behaviour of the car. The control on the orientation presents limitation caused by sensor inefficiency and low accuracy (the computation is referred to the column those are 8 in total) but in general it doesn't cause pratical problems.

5 Conclusions and Future Developments

The project successfully integrated a multi-zone Time-of-Flight sensor (VL53L5CX) with an STM32-based control architecture to implement a basic object tracking capability. The system demonstrates:

- Reliable initialization and configuration of the VL53L5CX sensor in 8×8 mode, including per-zone thresholding and continuous ranging.
- Interrupt-driven acquisition pattern that preserves real-time responsiveness while keeping processing out of the ISR.
- A straightforward per-column statistical approach to determine the most-likely target bearing and distance.
- A practical P-controller mapping distance error and lateral displacement to motor commands, including normalization, saturation and deadband logic for safe actuation.

This work demonstrates how a relatively modest hardware platform a low-cost STM32 NUCLEO board, a motor driver expansion and a modern multi-zone ToF sensor can be integrated into a coherent real-time system capable of perceiving and interacting with its environment. The project highlights the importance of careful low-level configuration (I^2C, interrupts, timer PWMs), robust handling of asynchronous events, and pragmatic control design.

Bibliography

1. STMicroelectronics, *VL53L5CX datasheets and application notes.* (Used for sensor reference and API).

2. Course slides, Prof. Luca De Cicco, *Mobile Robotics* — chapter on differential-drive kinematics and control (used for the mathematical model and control strategy).

3. STMicroelectronics, *X-NUCLEO-IHM04A1 datasheets and application notes.*