ERTC - Laboratory 2 - Open Loop Control

1 Introduction

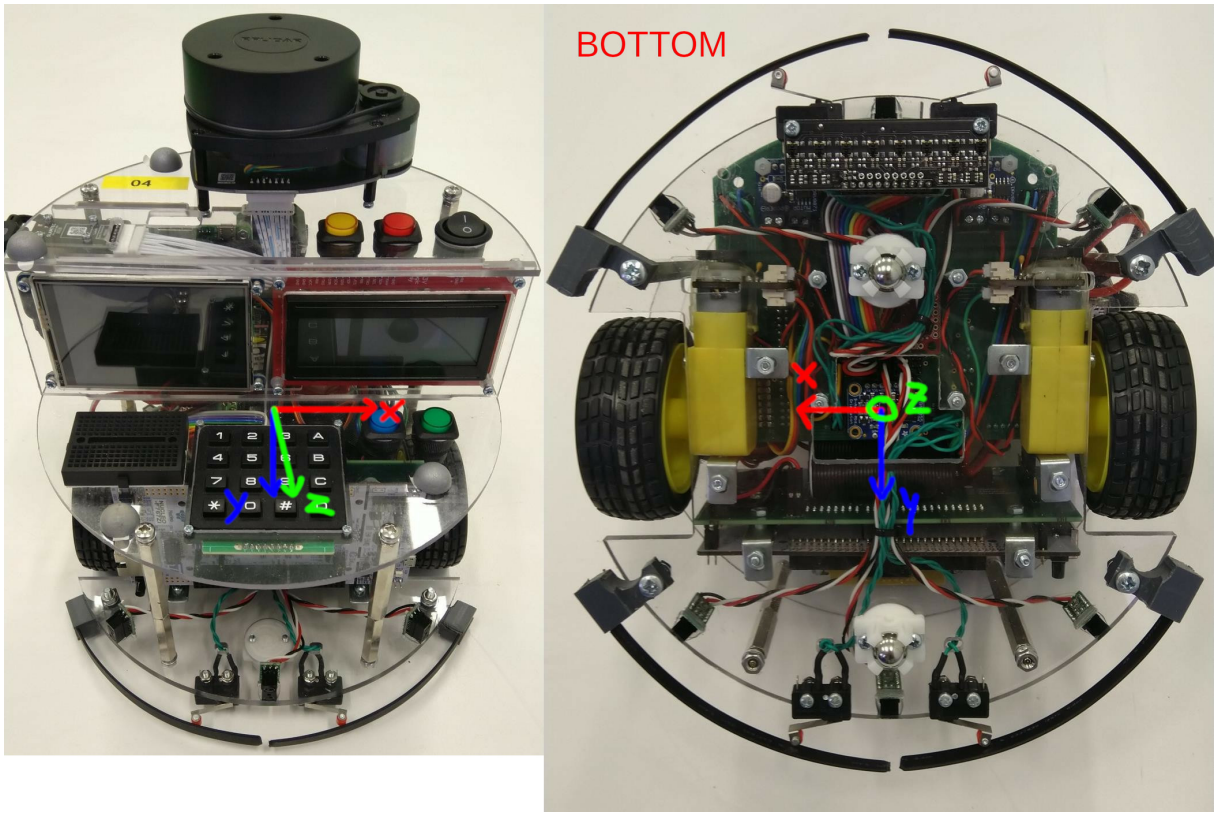
The purpose of this lab is to implement an open loop camera stabilizer using the IMU and servo motors.

2 Preliminary information

2.1 IMU

An IMU (Inertial Measurement Unit) is an electronic device that measures and reports a body's specific force, angular velocity, and orientation in space, using a combination of accelerometers, gyroscopes, and magnetometers. It is commonly used in robotics, drones, and other applications that require precise positioning and orientation information. The accelerometers measure linear acceleration, the gyroscopes measure angular velocity, and the magnetometers measure magnetic fields to determine the orientation of the device relative to the Earth's magnetic field.

On the TurtleBot, the axis of the accelerometer are aligned as in the figures below.



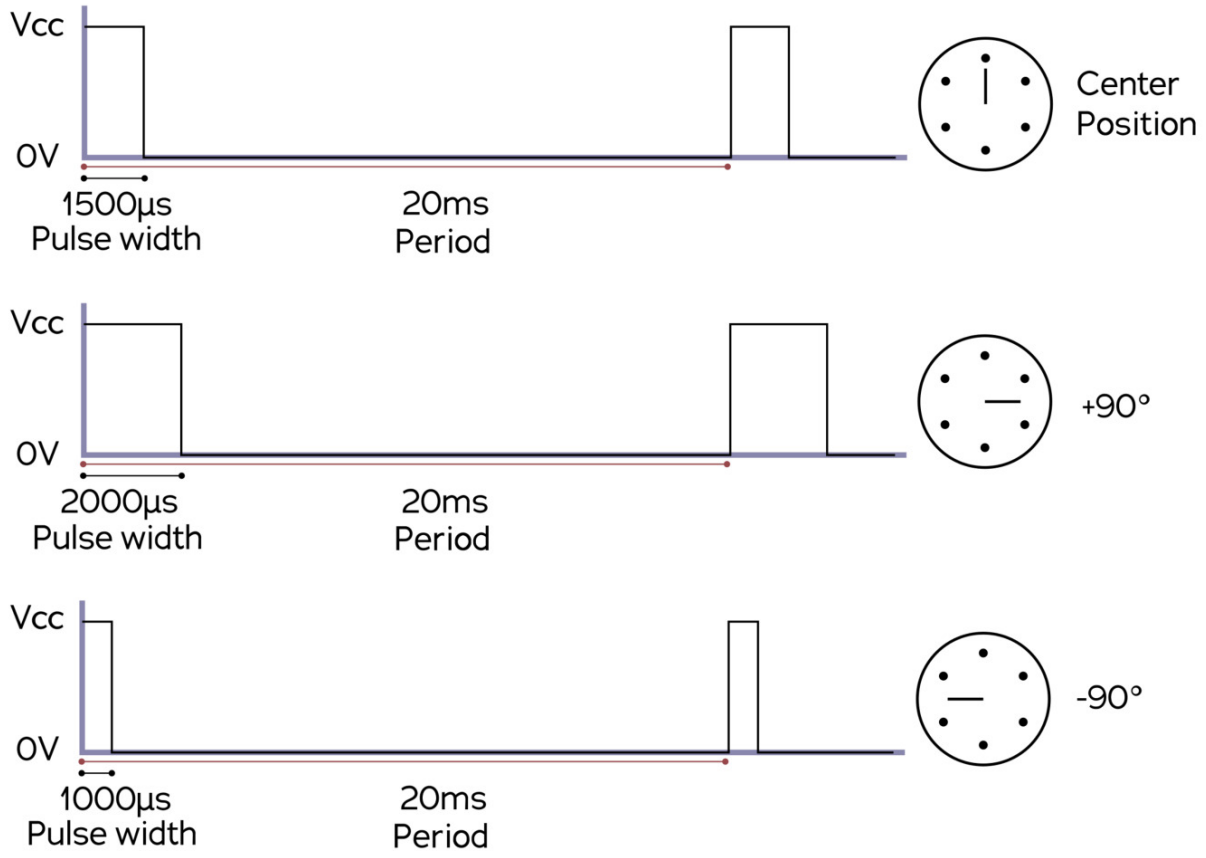
> **Note**: The ‘z’ axis is pointing downwards (toward the bottom of the robot)

2.2 Servo motors and camera

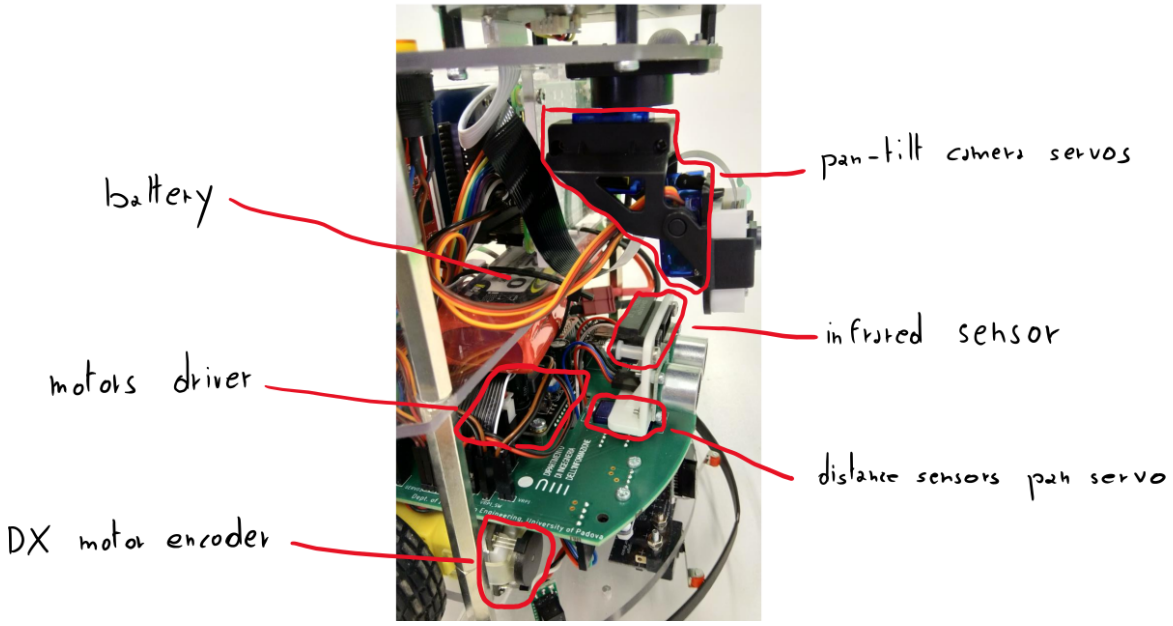
A servo motor is a type of motor that is designed to provide precise control over the rotation of its output shaft. It consists of a small DC motor, a gear train, and a feedback control system.

The internal feedback system is responsible for monitoring the actual position of the motor and comparing it to the desired position, then adjusting the motor’s speed and direction to achieve the desired position. This closed-loop feedback system allows servo motors to accurately and consistently control their output.

Servo motors are commonly controlled using pulse width modulation (PWM). To control a servo motor with PWM, a control signal is sent to the servo motor’s control circuitry, typically in the form of a series of pulses. The width of these pulses determines the position of the servo motor’s output shaft as shown in the following figure.



On the TurtleBot these are two servo that control the tilt and pan of a camera as shown in the figure below.



Specifically:

- Tilt refers to the vertical movement of the camera, where the camera is moved up or down.
- Pan refers to the horizontal movement of the camera, where the camera is rotated around its vertical axis to the left or right.

2.3 Finding the tilt of the Tbot

To find the current tilt of the Turtlebot we can use the measurement of acceleration on the **y** axis of the accelerometer you can use the equation:

$$\theta = sin^{-1}(a_y/g)$$

where θ is the tilt angle, a_y is the acceleration measured on the y axis and $g = 9.81$ is the gravity acceleration.

2.4 Moving the camera

Due to mechanical constraints we already configured the all the relevant parameters on the STM32 (i.e. PWM pins, PWM frequency, min and max servo angles) to move the servos without damaging them.

The PWM is generated using TIM1. In particular the tilt is controlled by CHANNEL_2 of TIM1, while the pan is controlled by CHANNEL_3. The core functions to move the camera are the following:

```
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_3,
                      (uint32_t)saturate((150+pan*(50.0/45.0)),
                         SERVO_MIN_VALUE, SERVO_MAX_VALUE)); // tilt

__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2,
                      (uint32_t)saturate((150+tilt*(50.0/45.0)),
                         SERVO_MIN_VALUE, SERVO_MAX_VALUE)); // pan
```

`__HAL_TIM_SET_COMPARE` is a HAL library macro used for setting the compare value of a specific channel of a hardware timer on a microcontroller. In this case, it is used to set the compare value for channels 2 and 3 of the TIM1 hardware timer.

The compare value determines the point in time at which an output signal will transition from a high to a low state, or vice versa, in relation to the timer's counter value.

The first argument, `&htim1`, is a pointer to a HAL TIM (Timer) handle, which is a data structure containing all the necessary information to operate the timer hardware.

The second argument, `TIM_CHANNEL_3` or `TIM_CHANNEL_2`, specifies which channel of the timer is being configured. In this case, channel 3 is being used to control the tilt of a device, and channel 2 is being used to control the pan.

The third argument is a calculated value, obtained by applying a formula that takes the `pan` or `tilt` angle as input and scales it using a certain factor. The result gives a PWM duty cycles from approximately 3.3% to 11.7% with a neutral position at 7.5% (assuming a frequency of 50 Hz).

The `saturate()` function is used to limit the PWM duty cycle to a specific range, specified by the `SERVO_MIN_VALUE` and `SERVO_MAX_VALUE` constants. This prevents the PWM signal from exceeding the limits of the servo motor or other hardware being controlled by the PWM signal.

The sample code includes all the required functions for adjusting the camera's position. The variables `tilt` and `pan` are utilized to control the camera's movement, with values assigned to each variable representing the desired change in orientation in degree ranging from **-45** to **+45** degree. Upon being assigned values, the servo motors will adjust the camera's tilt and pan angles accordingly.

2.5 Data logger

The TBot is equipped with a data logger that can records almost any type of data. There are available two communication interfaces: * Serial Datalogger (via USB dubug interface) * WiFi Datalogger

2.5.1 MATLAB

The serial datalogger can be invoked from MATLAB using the command:

```
data = serial_datalog(COM_port , packet_specification)
```

where `COM_port` is the port of the serial datalogger (e.g. COM3) and `packet_specification` is a cell array of char arrays that specifies the type of data to be recorded. The port currently in use can be immediately detected by invoking the `seriallist` routine from the Matlab Command, provided that the STM32F7 board is the only device connected to the host computer. If so, the routine should return a list of two items, of which the first is the COM1 port reserved by the system for internal uses, and the other is the port used by the STM32F7 board to communicate with the host computer.

The data received from the robot are simultaneously displayed on a Matlab figure in real–time, and saved into a memory buffer. Once the figure is closed, the buffer content is copied into the output variable data, which is a Matlab structure with the following fields:

- **data.time** is the field containing the time instants at which the data samples have been transmitted, with respect to the beginning of the data logging process.
- **data.out** is a cell array containing the data samples sent by the balancing robot. Each element of the cell array corresponds to a different logged signal. It is a matrix, whose columns are the values of the logged signal. Therefore, the number of rows is equal to the size of the signal (number of components), and the number of columns is instead equal to the number of received data samples. The number of received data samples never exceeds the size of the buffer used to hold the data during the data logging process (by default, the buffer size is equal to 1000). For example, the command y = data.out{2}(2,:) retrieves the 2nd component of the 2nd signal sent by the STM32F7 board.

For example, by invoking the command:

```
data = serial_datalog('COM8',{'2*single','2*single'}, 'baudrate',
    115200)
```

the data logger will expect to receive a total of four single precision floating point numbers, that will be displayed in two separate sub-plots.

To use the wifi datalogger, the following command should be invoked:

```
data = udp_datalog(ip, port, packet_specification)
```

where `ip` is the IP address of the robot, `port` is the port number used by the wifi datalogger, and `packet_specification` is a cell_array of char array that specifies the type of data to be recorded.

For example, you can invoke the command

```
data = udp_datalog('147.162.118.61',9090, {'2*single','2*single'})
```

ATTENTION: every turtlebot has a different IP address. Please check (or ask for it) the IP address of the robot you are trying to connect to.

If something hangs or crash during the data logging process and nothing is working anymore, you can try those two commands:

```
clear all
instrreset
```

2.5.2 STM32

On the STM32 side, the datalogger has to be properly configured. The example is provided in the project for the lab 2.

You can choose which communication interface you want to use by calling the following instructions inside the main function (those function calls are already included in the project):

```
/* Configure data logger  */
logger.uart_handle = huart3; // for serial
//logger.uart_handle = huart2; // for wifi
```

The basic steps that allow to initialize and to run the data logger are explained in the code below:

```
/* declare an ertc_dlog struct */
struct ertc_dlog logger;
.
.
.
int main(void)
{
    .
    .
    .
    /* USER CODE BEGIN 2 */
    /* select the correct uart : uart3 is the default peripheral used
       */
    logger.uart_handle = huart3;
    // logger.uart_handle = huart2; // for wifi
    /* USER CODE END 2 */
    .
    .
    .
    while (1)
    {
        /* USER CODE BEGIN 3 */
```

```
22          logger_data.w1 = 10;
23          logger_data.w2 += 1.085;
24          logger_data.u1 = -3.14;
25          logger_data.u2 = 0.555683;
26          ertc_dlog_update(&logger); // check if someone is connected and
                waiting for data
27          ertc_dlog_send(&logger, &logger_data, sizeof(logger_data)); //
                send data
28      }
29      /* USER CODE END 3 */
30  }
```

To send arbitrary data it is necessary to define a struct this way:

```
1  struct datalog
2  {
3      float w1, w2;
4      float u1, u2;
5  } logger_data;
```

The structure above is just an example; the structure can be arbitrary, but it is important to put the datalog attribute after the word struct.

> **WARNING**: To avoid excessive transmission time and delays please send only numeric data (e.g. float, int, uint8_t, etc.). Do **not** send strings.

3 Warnings

Due to some incompatibilities issues, don't use the *printf* function. Instead, you can send, for example, a *uint8_t* value representing the state of the robot using the datalogger and visualize it in MATLAB

4 Exercises

4.1 Exercise 1

Based on the given information, it is required to develop a control law to stabilize the camera's tilt by utilizing data from an IMU. Basically, rotating the TBot along the x axis, we want to keep the camera aligned (0 degree) with the horizon. The IMU measurements are being read at a frequency of 100Hz, and the example code already implements the acquisition of IMU measurements. To extract the measurements, the accelerometer and gyroscope data can be obtained using the functions

`bno055_convert_double_accel_xyz_msq()` and `bno055_convert_double_gyro_xyz_rps()`, respectively. Both functions require a pointer to a structure as input, which is already defined in the example code as `d_accel_xyz` and `d_gyro_xyz`.

The declaration and definition of the above-mentioned functions are provided in the file `bno055.h` and `bno055.c`.

Modify the structure `datalog` to include the accelerometer data and the computed control signal.

Plot the data in MATLAB and compare the measurements with the control signal. **Plots of the acquired signals should be included in the final report.**

5 Bonus

5.1 Bonus 1

Implement a control algorithm that implement a "*smooth pan*" control (i.e. a control system that compensate sharp horizontal rotations of the TBot) using data from the gyro.

5.2 Bonus 2

Improve the control law using the other axis of the accelerometer