Hochschule Karlsruhe
University of
Applied Sciences

HKA

VGU
Vietnamese - German University

MICROCONTROLLER PROJECT REPORT

ROBOTICS ARM

Lecturer: Master, Lab Engineer Nguyen Hoang Vinh Khang

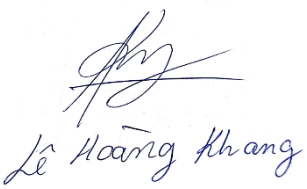
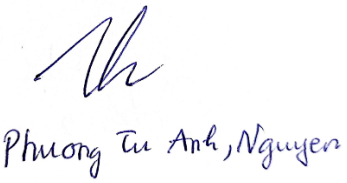
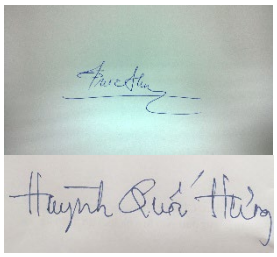
Team members:

Student 1: Le Hoang Khang _ **Student ID:** 20423010

Student 2: Nguyen Phuong Tu Anh_**Student ID:** 20423005

Student 3: Huynh Quoc Hung **Student ID:** 20422002

Intake: MST2023

Signature		
		

August 20th, 2024

1. Introduction

This project aims to develop a 3-degree-of-freedom robotic arm, designed for object manipulation using RTOS and some basic embedded techniques, as a requirement for completing the microcontroller lab course.

The robot arm is powered by stepper motors with TB660 drivers for each joint and features a gripper operated by a servo. The project includes an ultrasonic sensor, HCSR04, for object detection and measurement. The system is controlled by an STM32F4 Discovery Board microcontroller which also integrates a microphone module, which could be used for voice command control in hands-free operations. Inverse kinematics is implemented using the geometric method for its simplicity and straightforwardness.

2. Objectives of this project

Demonstrate the usage of:

- RTOS (FreeRTOS),
- Kalman Filter,
- Embedded Machine Learning (Edge Impulse),
- Fuzzy Logic

3. Time schedule for this project:

Time	Work
June 26th	Borrowing robotic arm model of Mr. Tai
June 27th – July 11th	Controlling the stepper motor using the motor drive and opto to protect the STM32F407 discovery board
12th – 17th July	Applying forward kinematic and inverse kinematic to calculating the degree of the three main joints on robotic arm in order to establish the algorithm to control three main degrees
	Studying about the principle of Fuzzy Logic in adjusting the data of signals transmitted from microcontroller to real models
12th – 31th July	Adding the algorithm established into FREETOS to locate the gripper moving to the desired location
17th – 31st July	Studying about the principle of ultrasonic sensor HCRS04 and configuring the CubeMX as well as programming on Keil C to measure the distance from object to robotic arms
1st – 7th August	Using Kalman Filter in filtering the noise of the ultrasonic wave signal transmitted and received from the ultrasonic sensor to object and vice versa in order to gain the accuracy result.
8th August	Adding the void function of measuring the distance from an object to ultrasonic sensor into the forward and inverse

	kinematic algorithm to determine the degree of three main joints
8th – 10th August	Adding FreeRTOS to determining the degree of the three main joints
20th July – 24th August	Studying about the principle of machine learning - Collecting the audio signal samples on Edge Impulse and export the file which can be included on Keil C - Using Keil C to record the voice through MEMS on STM32F407 board

4. Hardware

- 01 x Robotic arm
- 03 x driver stepper motor
- 03 x TB6600
- 01 x gripper
- 01 x servo RC MG996 R
- 01 x STM32F407 board
- 01 x ultrasonic sensor HCSR04

5. Principle of robot arm control

5.1. Principle of the movement of robotics arm

5.1.1. Forward kinematic and inverse kinematic

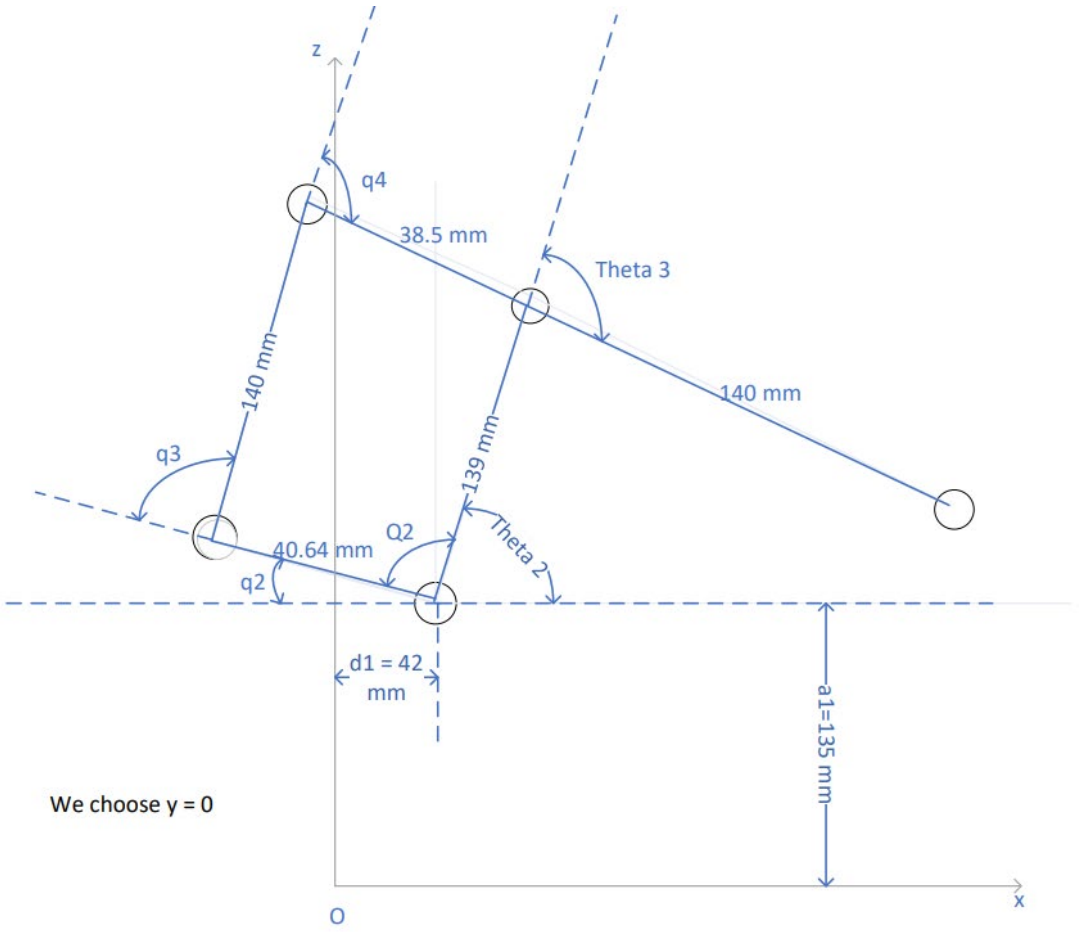


Figure 1: Parallel Mechanism of Forward and Inverse Kinematics of Robotic Arm [3]

Take a look at the parallel mechanism [3], we can use the same equations to calculate *theta2* and *theta3* as they are *q1* and *q2* in the last section. Thus, we have:

$$theta3 = -\cos^{-1}\frac{x^2+y^2-140^2-139^2}{2.140.139} \text{ (with y = 0)} \quad (1)$$

$$theta2 = \tan^{-1}\frac{y}{x} + \tan^{-1}\frac{140.\sin(theta3)}{139+140.\cos(theta3)} \text{ (with y = 0)} \quad (2)$$

$$Q_2 = \pi - theta2 \quad (3)$$

$$q2 = theta3 - theta2 \quad (4)$$

5.1.2. Motor stepper

Stepper motor drivers are small pin-compatible modules designed particularly to drive stepper motors capable of continuously rotating with precise position control without the need for a feedback system. The stepper motor drivers have built-in converters that offer multi-stage resolution and variable current control that allows control of stepper

motors with simple step and direction inputs [1]. In this article, the TB6600 motor driver is used. This driver IC is easy to use, effective in control and compatible with multiple devices such as Arduino and STM32F4 series. Here are technical specifications of TB6600:

- The operating voltage ranges from 9-40V DC.
- The output current of the module is 0.7A-4.0A and it is selected in 8 steps through DIP switches
- The input pulse frequency is up to 20KHz.
- Input signal suitable for – 5V signal levels.
- Pulse per revolution is 200-6400.
- The logic signal current is 8A-15A.
- It is suitable for 2-phase and 4-phase stepper motors.
- It provides protection from overcurrent overheating.
- Inputs are isolated optically.
- Insulation resistance is 500 megohms.
- It can support PUL/FIR mode.

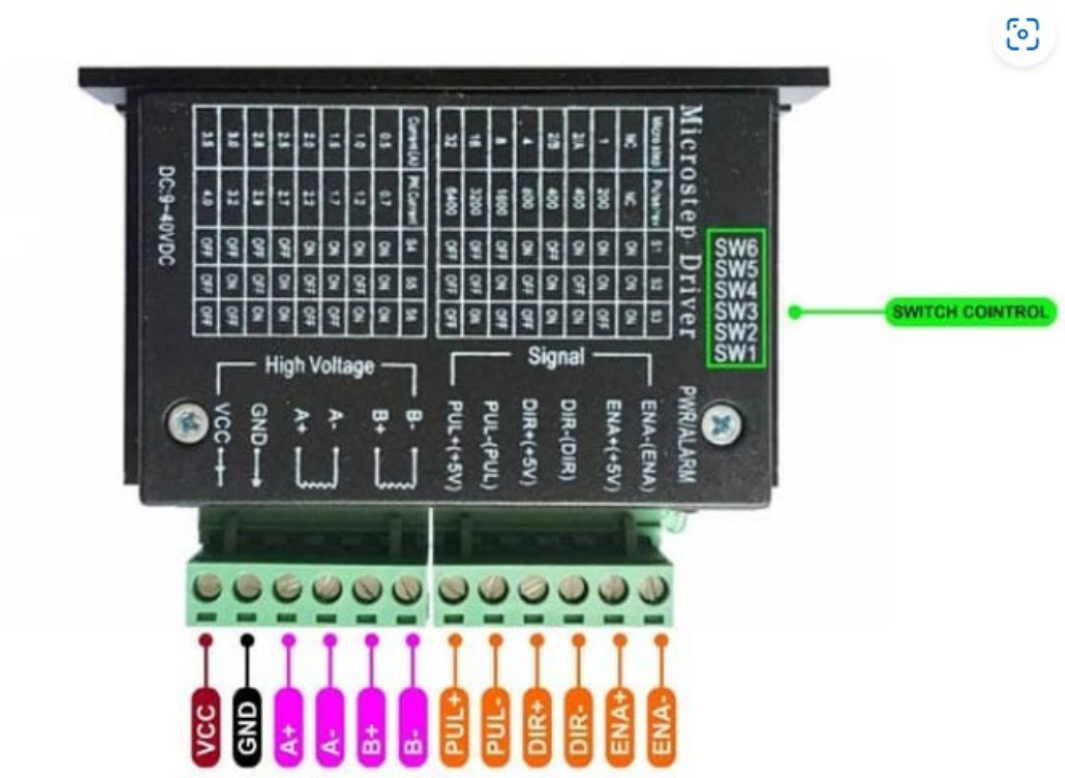


Figure 2: TB6600 Stepper Motor Driver 4A Module

For more information about this module, there is a tutorial website for this: [TB6600 Stepper Motor Driver : Datasheet & Its Applications (watelectronics.com)]

5.2. Principle of ultrasonic sensors for object detection and distance measurement

In this project, we aim to use 2 HC_SR04 ultrasonic sensors for the object detection part. HC_SR04 ultrasonic sensors use sonar to determine the distance to an object. It is compatible with many type of microcontroller unit (MCU) such as Arduino, EMP32 and STM32. With the easy-to-use interfaces and its accuracy, HC SR04 sensor is choose for this project. Here are technical specifications of the sensor [2]:

- Operating voltage: +5V
- Theoretical Measuring Distance: 2cm to 450cm
- Practical Measuring Distance: 2cm to 80cm
- Accuracy: 3mm
- Measuring angle covered: <15°
- Operating Current: <15mA
- Operating Frequency: 40Hz

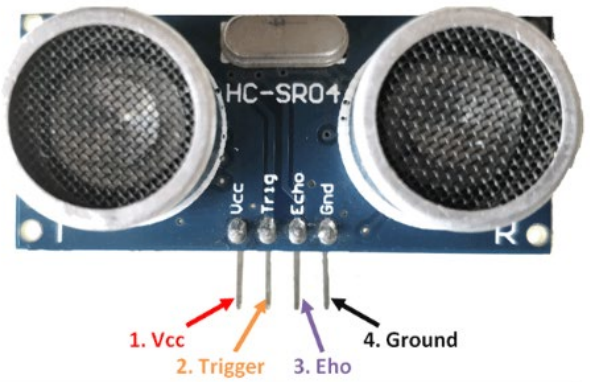
The sensor is placed opposite the robot arm with 30 centimeters. However, due to the fluctuation of the output, a Kalman filter is applied to the output data of the sensor to reduce the fluctuation of the data. In this program, 2 formula is presented to calculate the distance from the object to the robot arm:

In x direction, the distance from the robot arm to object is calculated as:

$$x = 300 - input\ data\ of\ the\ sensor$$

In y and z directions, the distances are constant with 0 and 50 mm for y and z respectively. Note that all the parameters is represented as millimeters.

HC-SR04 Ultrasonic Sensor Module



HC-SR04 Ultrasonic Sensor Pinout

Figure 3 HCSR04 ultrasonic sensor and pinout.

5.3. Principle of audio recording on MEMS of STM32F407 board

The principle of the audio recording on MEMS of STM32F407 is based on the example and instruction of STM company. The microphone to collect data is MEMES MP45DT02 module, which is applied onboard of STM32F4 discovery. This module converts sound waves into electrical signals through its vibrating diaphragm. These signals (PDM) are then amplified and digitized to PCM for further processing. Here are some technical specifications:

- Single supply voltage
- Low power consumption
- 120 dBSPL acoustic overload point
- Omnidirectional sensitivity
- PDM single-bit output with option for stereo configuration
- HLGA package (SMD-compliant) plastic or metal
- ECOPACK® , RoHS, and "Green" compliant

For audio playback, STM32F4 board uses its DAC – module CS43L22 (U3) to converted digital audio back into an analog signal. MEMS components ensure the playback is clear and accurate, preserving the quality of the recorded audio. Here are technical specifications of module U3:

- 98 dB Dynamic Range (A-wtd)
- 88 dB THD+N
- Headphone Amplifier – GND Centered
  - No DC-Blocking Capacitors Required
  - Integrated Negative Voltage Regulator
  - 2 x 23 mW into Stereo 16 Ω @ 1.8 V
  - 2 x 44 mW into Stereo 16 Ω @ 2.5V
- Stereo Analog Input Passthrough Architecture
  - Analog Input Mixing
  - Analog Passthrough with Volume Control
- Digital Signal Processing Engine
  - Bass & Treble Tone Control, De-Emphasis
  - PCM Input w/Independent Vol Control
  - Master Digital Volume Control and Limiter
  - Soft-Ramp & Zero-Cross Transitions
- Programmable Peak-Detect and Limiter
- Beep Generator w/Full Tone Control

- Tone Selections Across Two Octaves
- Separate Volume Control
- Programmable On and Off Time Intervals
- Continuous, Periodic, One-Shot Beep Selections

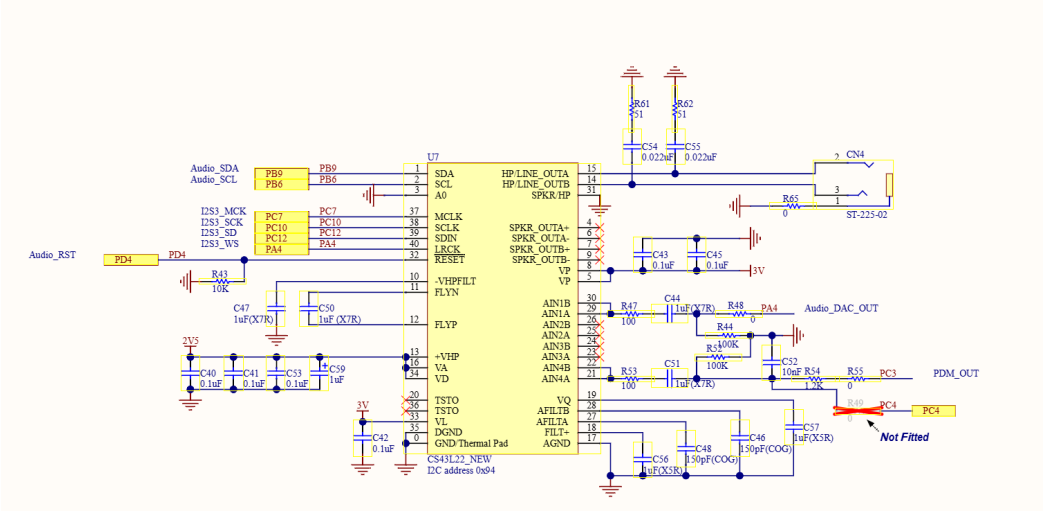


Figure 4 Module U3 – CS43L22 on STM32F4 discovery board

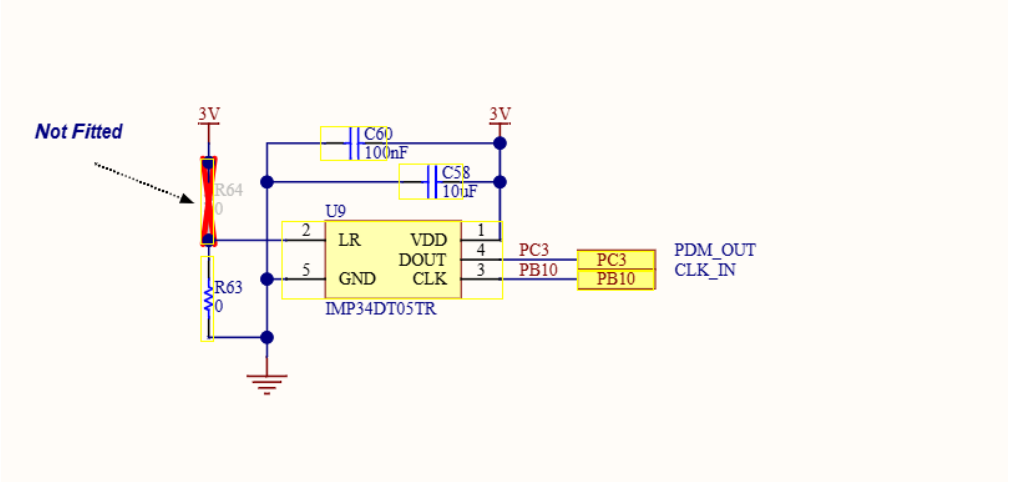


Figure 5 Module U9 – MP45DT02 on STM32F4 discovery board

For the detailed information about these 2 modules, please refer to the following samples from ST: MB997, AN5027, UM1721, MP45DT02.

5.4. Embedded Machine Learning in keywords spotting application

The steps to utilize the Edge Impulse machine learning framework [4] are as follows:

- Data Collection: Collect audio from various sources such as microphones or prebuilt datasets.

    - Key takeaways for data collection include:
        - Aim for at least 10 minutes of audio per class; the more data, the better.
        - In addition to collecting predefined keywords, it's also necessary to gather audio samples that don't match these keywords, such as background noise (e.g., running water) or other human speech (classified as "unknown"). This is crucial because a machine learning model learns solely from the data it is provided and doesn't inherently distinguish right from wrong. The more varied your data, the better the model's performance.
        - Ensure a balanced amount of data is collected for each audio class: keywords, noise, and unknown. Balanced datasets generally produce better outcomes.
        - Lat but not least, some keywords are more difficult to distinguish than others, particularly single-syllable words like "One," which can result in false positives (e.g., being mistaken for "Gone"). This is why companies like Apple, Google, and Amazon often use keywords with at least three syllables, such as "Hey Siri," "OK, Google," and "Alexa."
- Feature Extraction: Choose signal processing blocks from the Edge Impulse framework to extract key features, i.e. the most important pieces of information. This process simplifies raw data, which contains a lot of redundant information, into more representative features.
    - Important notes for feature extraction include:
        - MFCCs are effective in representing keywords because they condense complex acoustic information into a set of coefficients that emphasize the frequencies most important for human speech perception.
            - By capturing the critical aspects of speech relevant to human perception and filtering out unnecessary information, MFCCs enable machine learning algorithms to distinguish between different keywords with high accuracy.
- Neural Network Training: Pass these extracted features into a neural network block within the Edge Impulse framework. The network will learn to differentiate between various audio classes (keywords, or noise, or unknown).
    - Significant points for neural network training include:
        - Neural networks are algorithms, loosely modeled after the human brain, that can learn to recognize patterns in the training data. The network we're training takes the features produced by the MFCC as input and maps them to one of the predefined classes: keywords, noise, or unknown.

- The input voice is processed through intermediate layers of the neural network, ultimately producing two values: the probability that the input represents your keyword, and the probability that it represents noise or unknown.

- Code Generation and Integration: Finally, generate the code with the keywords classification capabilities and integrate it into our target embedded device.

5.5. Fuzzy Logic Control for Servo Motor Speed Based on Angle and Load

Fuzzy logic is a mathematical approach to reason based on degrees of truth, rather than the traditional Boolean logic where a statement is either true or false. This allows for more comlpex and flexible decision-making, especially in situations where there is uncertainty or ambiguity.

Key Concepts:

- **Membership Functions:** These functions define the degree to which an element belongs to a set. For example, a membership function for "tall" might assign a degree of 0.8 to a person who is 1.8 m tall and 0.3 to a person who got 1.6 m in height.
- **Fuzzy Sets:** These are sets where elements can have varying degrees of membership. For instance, a fuzzy set of "tall people" might include individuals of different heights with varying degrees of membership.
- **Fuzzy Rules:** These are rules that relate fuzzy sets and their degrees of membership. For example, a rule might state: "If temperature is high and humidity is high, then the likelihood of rain is high."
- **Fuzzy Inference:** This is the process of drawing conclusions from fuzzy rules and fuzzy inputs. It involves combining the degrees of membership of the inputs and applying the rules to arrive at a conclusion.

In our project, we think we could use fuzzy logic in controlling the speed of the servo motor with the following assumptions:

- Small angle is from 0 to 120 degree
- Mild angle is from 120 to 240 degree
- Large angle is from 240 to 360 degree
- Light load is from 0 to 0.1 kg
- Medium load is from 0.1 to 0.2 kg
- Heavy load is from 0.2 kg up

The following are a potential fuzzy logic control settings corresponding to angle and load:

- **Fuzzy Rules:**
  1. If angle is small and load is light, then speed is slow.
  2. If angle is medium and load is light, then speed is medium.
  3. If angle is large and load is light, then speed is fast.
  4. If angle is small and load is medium, then speed is medium.
  5. If angle is medium and load is medium, then speed is medium.
  6. If angle is large and load is medium, then speed is fast.
  7. If angle is small and load is heavy, then speed is slow.
  8. If angle is medium and load is heavy, then speed is slow.
  9. If angle is large and load is heavy, then speed is medium.
- **Membership Functions:**
  - **Angle:**
    - Small: Triangular membership function with peak at 60 degrees, left edge at 0 degrees, right edge at 120 degrees.
    - Medium: Triangular membership function with peak at 180 degrees, left edge at 120 degrees, right edge at 240 degrees.
    - Large: Triangular membership function with peak at 300 degrees, left edge at 240 degrees, right edge at 360 degrees.
  - **Load:**
    - Light: Triangular membership function with peak at 0.05 kg, left edge at 0 kg, right edge at 0.1 kg.
    - Medium: Triangular membership function with peak at 0.15 kg, left edge at 0.1 kg, right edge at 0.2 kg.
    - Heavy: Triangular membership function with peak at 0.25 kg, left edge at 0.2 kg, right edge at 0.3 kg.
  - **Speed:**
    - Slow: Triangular membership function with peak at 0.2 (normalized speed), left edge at 0, right edge at 0.4.

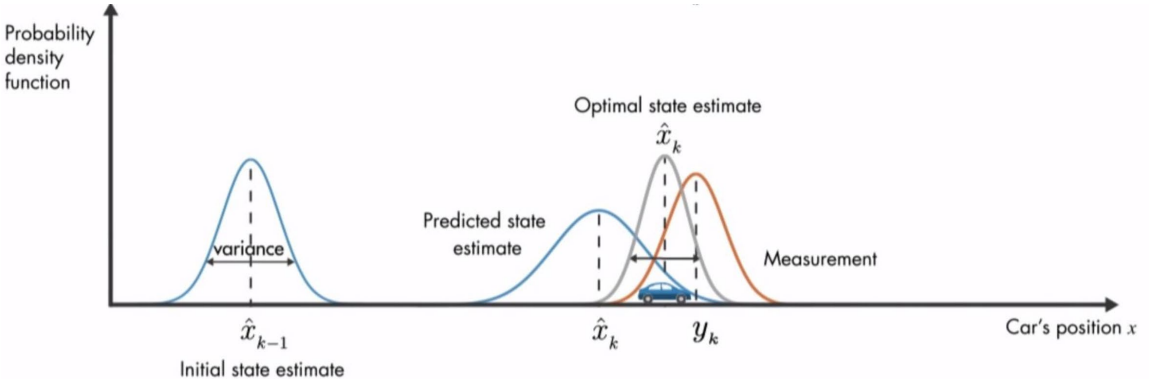
    - Medium: Triangular membership function with peak at 0.6 (normalized speed), left edge at 0.4, right edge at 0.8.
    - Fast: Triangular membership function with peak at 1 (normalized speed), left edge at 0.8, right edge at 1.

- **Fuzzy Inference:**
    - **Scenario:** For example, the desired angle is 250 degrees, and the load is measured to be 0.18 kg.
    - **Fuzzification step:**
        - Angle: 0.6 (medium), 0.4 (large)
        - Load: 0.6 (medium), 0.4 (heavy)
    - **Rule evaluation step:**
        - Rule 1: 0.6 * 0.2 = 0.12
        - Rule 2: 0.6 * 0.6 = 0.36
        - Rule 3: 0.4 * 0.2 = 0.08
        - Rule 4: 0.6 * 0.4 = 0.24
        - Rule 5: 0.6 * 0.6 = 0.36
        - Rule 6: 0.4 * 0.2 = 0.08
        - Rule 7: 0.6 * 0.4 = 0.24
        - Rule 8: 0.6 * 0.6 = 0.36
        - Rule 9: 0.4 * 0.4 = 0.16
    - **Aggregation (using maximum) step:**
        - Slow: 0.24
        - Medium: 0.36
        - Fast: 0.08
    - **Defuzzification (using centroid) step:**
        - Calculate the centroid of the fuzzy output set. In this case, the centroid is approximately 0.34.

Conclusion: The optimal normalized speed is determined to be around 0.34, which corresponds to a **medium** speed based on the defined membership functions. This means the servo motor should rotate at a speed that is roughly between the slow and medium speeds, considering the given angle and load.

5.6. Kalman Filters for Noise Reduction in Ultrasonic Distance Sensors

The Kalman Filter is a statistical technique used to estimate the state of a dynamic system. It's particularly useful in situations where direct measurement is noisy or incomplete. The filter combines predictions from a mathematical model with measurements to produce an estimate of the system's state that tends to be more accurate than either source of information alone [5].



Key Components:

- **Process Model**: This describes how the system evolves over time. It's often represented as a state equation that predicts the next state based on the current state and a process noise term.
- **Measurement Model**: This relates the system's state to the observed measurements. It's represented as an observation equation that models how the measurements are corrupted by noise.
- **Prediction Step**: Using the process model, the filter predicts the next state and its associated covariance matrix.
- **Update Step**: When a new measurement becomes available, the filter updates its estimate by combining the predicted state with the measurement. This involves calculating a Kalman gain, which determines how much weight to give to the measurement.

Kalman Filters for Noise Reduction in Ultrasonic Distance Sensors:

Ultrasonic distance sensors are widely used in various applications due to their ability to measure distances accurately. However, their measurements can be affected by

noise, which can degrade the accuracy and reliability of the sensor. **Kalman filters** provide an effective solution for reducing noise in ultrasonic distance sensor measurements.

Noise Sources in Ultrasonic Distance Sensors:

- **Echo Interference:** Multiple reflections of the ultrasonic wave can interfere with the desired echo, leading to erroneous measurements.
- **Environmental Factors:** Factors such as temperature, humidity, and air currents can affect the propagation of sound waves, causing measurement errors.
- **Sensor Noise:** The sensor itself may introduce noise due to electronic components or manufacturing imperfections.

How Kalman Filters Work for Noise Reduction:

1. **Process Model:** The Kalman filter models the dynamics of the distance sensor's measurements. In this case, the process model can be a **simple random walk**, assuming that the distance changes slowly over time.
2. **Measurement Model:** The measurement model relates the true distance to the noisy sensor measurements. It accounts for the sensor's noise characteristics, such as its variance.
3. **Prediction Step:** The Kalman filter predicts the next distance based on the previous state and the process model.
4. **Update Step:** When a new sensor measurement is available, the filter updates its estimate by combining the predicted state with the measurement, weighted by the Kalman gain. The Kalman gain determines how much to trust the measurement relative to the prediction.

Implementation Considerations:

- **Choosing the Right Process Model:** The choice of the process model depends on the specific application and the expected dynamics of the distance measurements.
- **Tuning the Kalman Gain:** The Kalman gain determines the trade-off between responsiveness to new measurements and stability. It can be tuned based on the noise characteristics and desired performance.
- **Handling Outliers:** Outliers can significantly affect the Kalman filter's performance. Techniques like outlier detection and rejection can be used to mitigate their impact.

Alternative Process Models for Ultrasonic Distance Sensors:

While the simple random walk model is often used for ultrasonic distance sensors, other process models can be considered depending on the specific application and noise characteristics. Here are some alternatives and their pros and cons:

1. Autoregressive (AR) Model

- **Pros:** Can capture more complex dynamics, especially for correlated noise.
- **Cons:** Requires estimating additional parameters (AR coefficients).

2. Moving Average (MA) Model

- **Pros:** Can model noise with short-term memory.
- **Cons:** May not capture long-term trends.

3. Autoregressive Moving Average (ARMA) Model

- **Pros:** Can capture both AR and MA dynamics.
- **Cons:** Requires estimating more parameters.

Factors to Consider When Choosing a Process Model:

- **Noise Characteristics:** Is the noise correlated or uncorrelated? Does it have short-term or long-term memory?
- **System Dynamics:** Are there any known trends or periodic patterns in the distance measurements?
- **Computational Complexity:** Consider the computational cost of estimating and using the model.
- **Performance Evaluation:** Experiment with different models and evaluate their performance using metrics like mean squared error or root mean squared error.

In general, the simple random walk model is a good starting point for many ultrasonic distance sensor applications. However, if you encounter significant noise correlation or more complex dynamics, exploring alternative process models may improve the Kalman filter's performance.

5.7. FREE RTOS applied in controlling robotics arm

In FreeRTOS, Semaphores and Mutexes are commonly used to manage shared resources and ensure proper parallel tasks synchronization, which is crucial in controlling a robotic arm.

- **Semaphores:** are used for signaling between tasks. They help coordinate tasks by ensuring that a task only runs when certain conditions are met, like waiting for a sensor signal or another task's completion. In our project, Motor2 should only start after Motor1 is done.
- **Mutexes:** can be used to protect shared resources. They prevent race conditions by ensuring that only one task can access the critical section at a time, such as accessing a shared I2C bus or a structure holding the state of the robotic arm. For instance, if Motor1 and Motor2 need to update the same variable representing the robotic arm's state, use a mutex to ensure only one task updates the variables at a time.

6. Implementation detail of each stage

6.1. Setup for controlling robotics arm model

To set up the hardware for controlling the 3-degree-of-freedom robotic arm, start by connecting the stepper motors to the TB6600 drivers. Identify the two coils of each stepper motor and connect them to the corresponding A+ & A-, B+ & B- terminals on the TB6600. Next, set the driver to operate at 1.5A current and configure the steps per revolution to 1600 pulses per revolution (ppr) using the dip switches on the TB6600. Then, connect a 24V DC power source to the driver's VCC and GND terminals, ensuring correct polarity.

For the control signals, connect the PUL+ (pulse) and DIR+ (direction) terminals of each TB6600 driver to the appropriate GPIO pins on the STM32F4 Discovery board, while the PUL- and DIR- terminals are connected to the ground (GND). After making these connections, link the enable signals if needed, and verify all connections for accuracy. Once everything is set, power on the system and test the stepper motor control to ensure the robotic arm movements are precise, with the specified current and step resolution. This setup will allow you to effectively control the arm using the STM32F4 Discovery board.

Table 1 GPIO connect from Driver to MCU

Driver TB6600		Pin
Stepper Motor Driver 1	PUL+	GPIOD9
	DIR+	GPIOD8
Stepper Motor Driver 2	PUL+	GPIOB15
	DIR+	GPIOB14
Stepper Motor Driver 3	PUL+	GPIOB13
	DIR+	GPIOB12

6.2. Controlling the stepper motor by microcontroller STM32F407

To set up the STM32F407 microcontroller for controlling the robotic arm, the first step is to configure the GPIO pins connected to the stepper motor drivers in output mode. This ensures the microcontroller can send the necessary pulse and direction signals to the

motors. For instance, GPIOD9 and GPIOD8 might be set as output pins for controlling the first motor. Following this, tasks are created in FreeRTOS to manage the motor control operations, as in Figure 6. Each task is responsible for controlling a specific joint, processing sensor inputs, and generating control signals. By using semaphores or mutexes indicate in Figure 7, these tasks are synchronized to ensure smooth and coordinated movements of the robotic arm, leveraging the real-time capabilities of FreeRTOS for efficient operation.

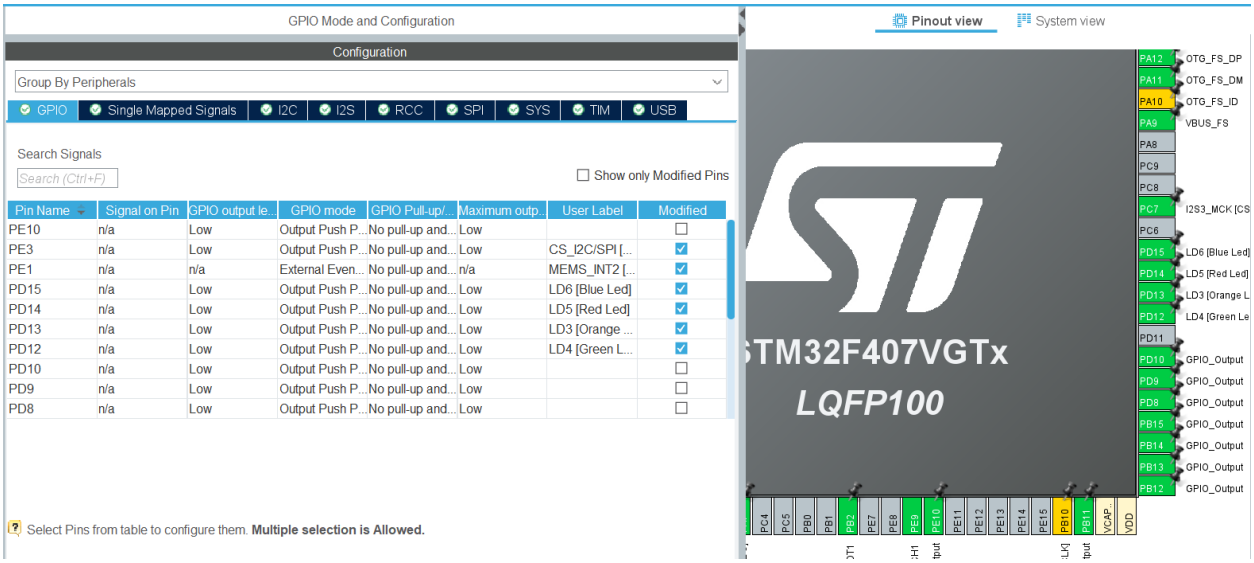


Figure 6 GPIO set up for motor driver TB6600 using Cube MX.

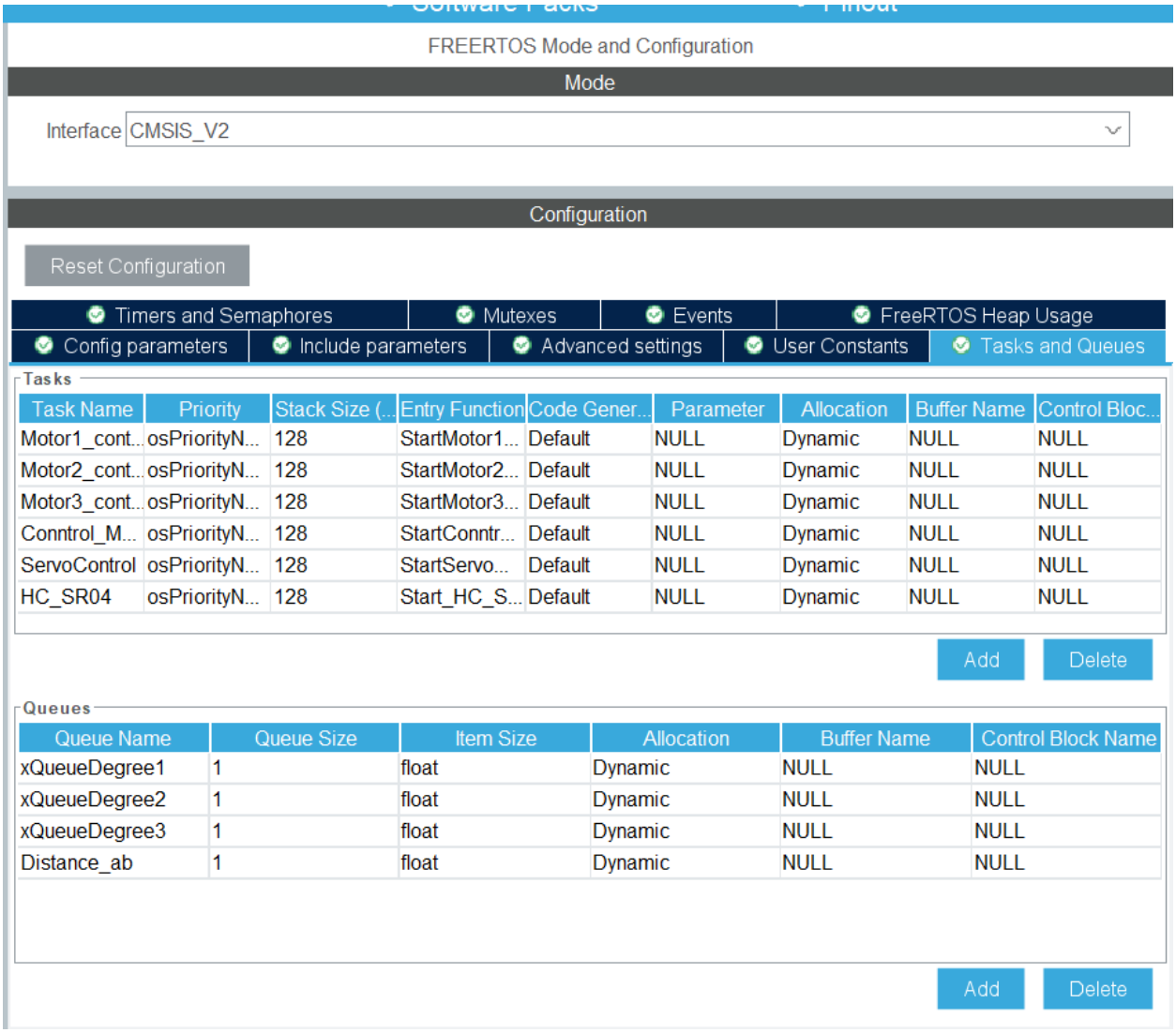


Figure 7. FreeRTOS setting tasks and queues for controlling motor using Cube MX.

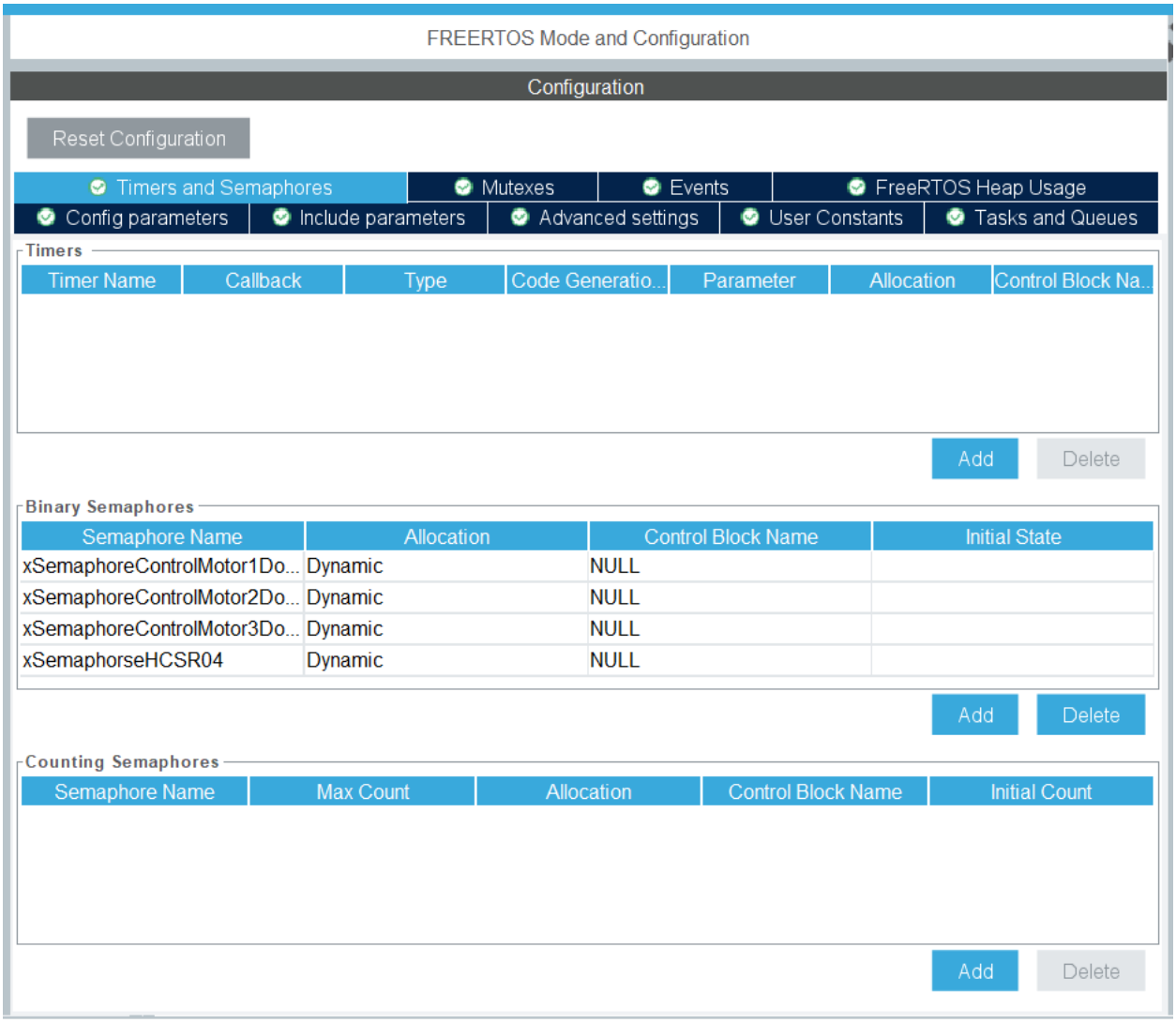


Figure 8 FreeRTOS setting for semaphores for controlling motor.

6.3. Controlling three main joints of robotics arm to move to the desired position by microcontroller STM32F407

After controlling a stepper motor from an initial degree to a desired degree, we control a robotic arm with 3 degrees of freedom (DOF) simultaneously. This program operates based on FreeRTOS, a real-time operating system, to manage multiple tasks (threads) running concurrently.

- *Configuration in CubeMX:* Timer 1 with Channel 1 and 2 (TIM1, CH1 and CH2) is triggered for adjusting the pulse width modulation (PWM) to control the changes of three degrees of freedom. Besides, GPIOs are configured as output channels to transmitting the signals to servo motor and stepper motors.

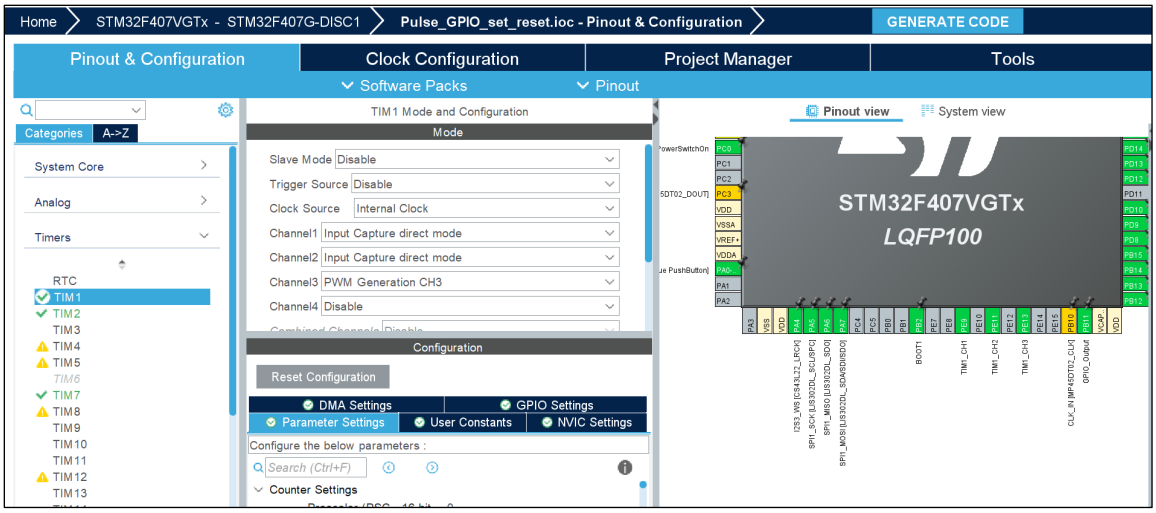


Figure 9: TIM1 Mode and Configuration

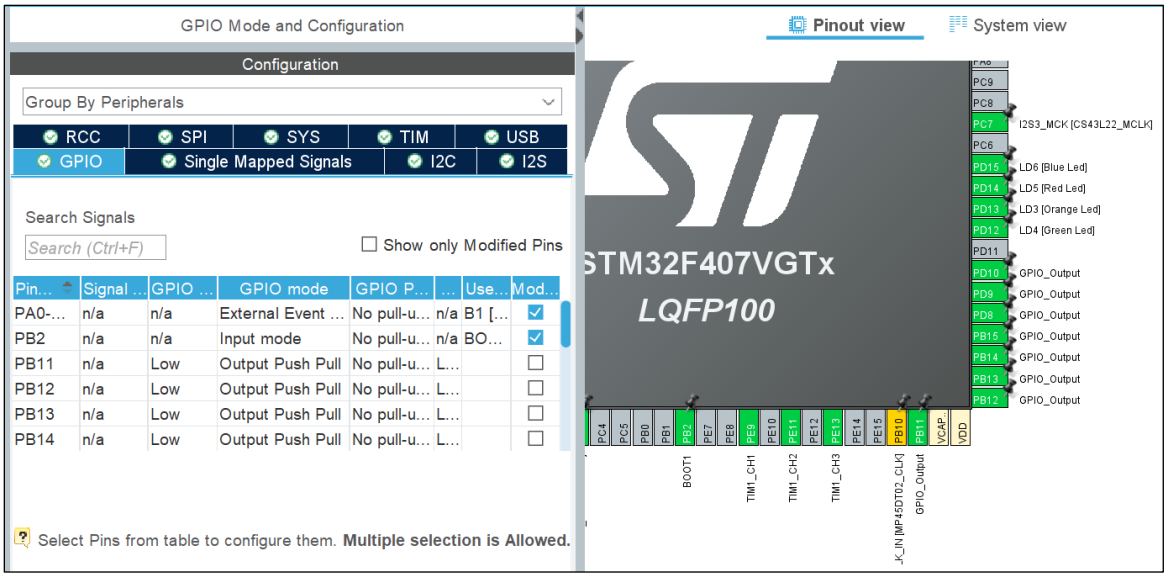


Figure 10: GPIOs configuration

- *Includes and Definitions:*

+ The header files (‘main.h’, ‘cmsis_os.h’) providing the definitions and declarations required for the STM32F407 hardware and the FreeRTOS system.

- *Private Variables:* FreeRTOS thread (task) and queue handles are also defined. These are used to manage tasks and inter-task communication, respectively.
- *Task Definitions: Motor Control Tasks* are used for controlling each of the three motors of the robotic arm (‘Motor1_control’, ‘Motor2_control’, and ‘Motor3_control’). Each task is responsible for controlling a specific motor, using

a stepper motor driver. *Control Motor Task* seems intended to manage the overall coordination of motor control, likely using feedback from sensors or other tasks.

- *RTOS Queues and Semaphores:*

+ *Message Queue:* Queues like 'xQueueDegree1', 'xQueueDegree2', 'xQueueDegree3', and 'Distance_ab' are used for passing messages (such as angles or distance measurements) between tasks.

+ *Semaphores:* Semaphores like 'xSemaphoreControlMotor1Done', 'aSemaphoreControlMotor2Done', and 'xSemaphoreControlMotor3Done' are used to synchronize tasks. For example, a motor control task may signal when a motor has reached its target position.

- *Task implementations:* Each of the task functions (e.g., 'StartMotor1_control', 'StartMotor2_control', etc.) is implemented as an infinite loop that will execute whenever the task is scheduled by the RTOS.
- *System Clock Configuration* is used to ensure the microcontroller operated at the correct frequency, which is crucial for timing-dependent tasks and peripheral operations.
- *Functions:*

+ 'osKernelStart()' is the RTOS scheduler started by creating the necessary threads and resources (queues and semaphores). This takes over control of the program, managing task execution according to their priorities.

+ Functions like 'HAL_TIM_PeriodElapsedCallback' are interrupt service routines (ISR) handling specific hardware events, such as a timer interrupt.

6.4. Measuring the distance from an object to robotics arm by HCSR04 ultrasonic sensor

Determining the distance from an object to robotics arm should be implemented to calculate the position of this object in 3D space coordination Oxyz. In this case, we just set the value of y equal to 0 because we put an object which we need to grasp on Ox axis to calculate the value of x. The reason we decided to set up one ultrasonic sensor is that we met some issues while using two ultrasonic sensors. While we applied two ultrasonic sensors at the same time, the interference between sensors easily occurs by the sound waves emitted by one sensor can be picked up by the other. As the result its may register the echo of the other sensor's pulse rather than its own. Besides, there are others difficulties we discuss in section 8. Regarding to the position of the gripper in the vertical axis (Oz), we defaulted it is approximately 50 mm to avoid this gripper from touching on the table surface which can cause the errors. In this section, we discuss about the way to measure the distance from an object to an ultrasonic sensor.

- *CubeMX Configuration:*

  + Timer 1 (TIM1) is used for input capture to measure the time it takes for the ultrasonic pulse to travel to the object and back. We triggered Channel 1 and Channel 2 of TIM1. We capture the rising edge through Channel 1 and the falling edge through Channel 2.

+ The GPIO pins are used as an output pin to trigger the ultrasonic sensor (TRIG pin and the echo pin).

- *Includes and Peripheral Declarations:*

+ '#include "main.h" and '#include "HC_SR04.h"' which is the library of HC_SR04 ultrasonic sensor.

- *In HC_SR04 library:*

  + *Global variables and definitions:*

- 'IC_Val1' and 'IC_Val2': These variables store the captured values of the timer when the echo signal is detected on the rising and falling edges, respectively.
- 'Difference' is the time difference between the rising edge and falling edge, representing the duration of the echo signal.
- 'Is_First_Captured' is the flag indicating whether the first edge (rising) has been captured.
- 'Distance' is the calculated distance to the object, based on the time difference.
- 'TRIG_PIN' and 'TRIG_PORT' are the GPIO pins and ports used for the TRIG signal of the HC_SR04 sensor.

+*'HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)':* is the function is triggered by an interrupt when the timer captures an input on channel 1 (which is connected to the ECHO pin of the HC_SR04 ultrasonic sensor).

- First Capture (Rising Edge): If 'Is_First_Captured' is 0 (indicating the first capture), it reads the timer value ('IC_Val1') when the rising edge is detected and changes the input capture polarity to detect the falling edge.
- If 'Is_First_Captured' is 1 (indicating the second capture), it reads the timer value ('IC_Val2') when the falling edge is detected, and then calculated the time difference between the two captured values. The function resets the flag 'Is_First_Captured' and reconfigures the input capture to detect the next rising edge.

$$\text{Distance} = \text{Difference} * 0.034 / 2 \qquad (5)$$

- Distance measurement function:

  + The TRIG pin is set high for 10 microseconds to send a pulse.

+ The TRIG pin is then pulled low, and the input capture interrupt is enabled to measure the echo duration.

+ The function returns the calculated ‘Distance’.

6.5. Recording audio signal through MEMS on STM32F407 [3]

Recording audio signal is implemented directly on Keil C, so that we determine the key components, libraries, key functions, callback, and I2S and DMA configuration.

- *Key components and libraries:*

+ ‘HAL_Init()’: Initializes the HAL library, configuring the system for further operations.

+ ‘SystemClock_Config()’ configures the system clock to run at 168 MHz.

+ ‘BSP_LED_Init()’ initializes the onboard LEDs.

+ ‘BSP_ACCELERO_Init()’ initializes the onboard MEMS accelerometer.

+ The libraries are compiled in this project in order to provide key functions and callbacks in the main codes to connect the MEMS microphone (MP45DT02) to the I2S peripheral in master receiver mode.

- *MEMS Microphone and I2S Handling:* This setup allows the microcontroller to receive audio data in PDM format. The recorded data is processed in 16-bit PCM format after conversion from PDM. This application uses a double-buffering mechanism to handle the audio data efficiently and prevent data loss during the recording process.
- *Audio Playback and Recording:* ‘COMMAND_AudioExecuteApplication()’ handles the commands for playing and recording audio based on user input.
- *Libraries:*
- *Wave Player library:* is stored in file ‘waveplayer.c’. This file contains the logic for playing audio files from a USB drive including key functions:
- ***WavePlayBack(uint32_t AudioFreq):*** reads audio data from the USB drive and sends it to the codec for playback.
- ***WavePlayerPauseResume(uint32_t wState):*** this function is declared to pause and resume the audio playback based on the provided state (‘PAUSE_STATUS’ or ‘RESUME_STATUS’).
- ***WavePlayerStop(void):*** this function is called to stop the audio playback and shuts down the audio output.
- ***WavePlayerInit(uint32_t AudioFreq):*** This function initializes the audio playback system, including the codec and all related peripherals (I2S, I2C, GPIOs).

- *Wave Recorder library:* The file of this library contains the logic for recording audio using the onboard MEMS microphone. The recorded audio is saved as a WAV file on the USB drive.
  - ***WaveRecorderStart():*** starts recording audio by initializing the I2S interface in master receiver mode to capture audio data from the MEMS microphone.
  - ***WaveRecorderStop():*** stops the audio recording and finalizes the WAV file.
  - ***WaveRecorderProcess():*** handles the main loop for audio recording, writing captured audio data to the USB drive and manages the recording duration and the user's input (like stop or pause).
  - ***WAV File Header Management:*** **'WavProcess_EncInit()', 'WavProcess_HeaderInit()',** and **'WaveProcess_HeaderUpdate()'** handle the creation and updating of the WAV file header, which contains metadata about the recorded audio.
  - *BSP library:* contains BSP callbacks for transfer audio signal data to the buffer data.
  - *FatFS library:* contains the functions to open and close the file, read and write the buffer data in the WAV file, create the file direction in the USB.
  - *PDM2PCM library:* consists of the functions and callbacks to transfer
- *Callbacks:*
  - *BSP_AUDIO_OUT_HalfTransfer_CallBack(void):* This callback is defined to handle the DMA half-transfer complete interrupt, signaling that the first half of the audio buffer has been transferred.
  - *BSP_AUDIO_OUT_TransferComplete_CallBack(void):* It manages the double-buffering mechanism for audio playback, ensuring smooth and continuous audio output.
  - *BSP_AUDIO_IN_TransferComplete_CallBack() and BSP_AUDIO_IN_HalfTransfer_CallBack()* manage the double-buffering mechanism for audio recording, ensuring that no audio data is lost during the recording process.
- *I2S and DMA Configuration:* is used to communicate with the audio codec and MEMS microphone. The audio data is transferred using DMA (Direct Memory Access) to ensure efficient handling of large amounts of data without burdening the CPU.

6.6. Embedded Machine Learning Code – Built by Edge Impulse

Project workspace: https://studio.edgeimpulse.com/studio/431644

Latest build: VGU_MST2023_uC_RobotArm_202408.1.0.5.pack

Detail guideline on how to use it: https://docs.edgeimpulse.com/docs/run-inference/using-cubeai

7. Our achievements

After nearly two months for us to struggle with the challenges we met in our project, we can control the degree of freedom (DOF) of three main joints on robotic arm move to the desired position. Besides, we can add the code used in controlling robotic arm in FreeRTOS. Regarding to machine learning part in our project, we have collected audio signal samples on Edge Impulse and export the file which can be included in Keil C to add the code controlling the DOF of three main joints, the servo motor, and stepper motors. Moreover, we can record and play the voice through the MEMS on STM32F407 board.

8. The difficulties we met during implementing our project

We did not know how to apply the Fuzzy logic by using encoder in our project to adjust three DOF move the accurate desired position. Besides, we have not found the way how to add the accelerometer to move the joints faster for the larger degree or more slowly for the smaller degree. We have not completed Machine Learning part because could not complied the audio recording and playing code, which was built on Keil C without the support of CubeMx with controlling robotic arm by FreeRTOS library, which was created by CubeMX. Because the FreeRTOS library is created on CubeMX have some different structures of functions and callbacks from the FreeRTOS library created for the audio recording and playing.

9. What we need to study and do more to fulfill our project's objectives

- For Fuzzy Logic, we need to create an encoder system on the hardware. In particularly, we have to change the configure of the external components related to hardware such as attaching an encoder system on the stepper motor directly.
- For Machine Learning, we have to modify the Free RTOS library created by CubeMX until it is suitable for the Keil C programming platform.

10. Future improvements and development of this project in the future

We think we need to develop our robotic arm project more professionally. In detail, we think we need to complete the Machine Learning system to give the commands for robotic arm run by human voice (eg. When we say "GO", "ON", "OFF", the microcontroller can receive the audio signal transmitted to the MEMS on STM32F407 board through fixing the errors we met in voice recognition by machine learning. More than that, we hope we can change the velocity of each joints when their degrees are modified.

11. Conclusion

Thank for the opportunities to do the robotic arm project with Mr. Nguyen Hoang Vinh Khang and nearly two months working in a group. Through exchange about the difficulties we met with the lecturer and our team, we can understand about the importance of microcontroller in sensor system. More than that, we were taught how to approach any issues we met such as the ways of reading the documents and how to configure CubeMX based on the documents. Besides, we learnt how to construct algorithms to collect signal data from one sensor or one sensor system. By this way, we can learnt how to use voice recognition and user button on STM32F407 board to give the commands for the robotic arm.

12. References

[1]. https://www.watelectronics.com/tb6600-stepper-motor-driver-module/HC-SR04 Ultrasonic Sensor Working, Pinout, Features & Datasheet (components101.com)

[2]. Le Anh Tai (2018). Bachelor Thesis: *Controlling a robotic arm using forward and inverse kinematic.* Vietnamese-German University. Department of Electrical Engineering & Information Technology Frankfurt

[3]. Nguyen Vu Hung. STM32_FW_F4/Projects/STM32F4-Discovery / Applications Audio/ Audio_playback_and_record. Website: https://gihub.com

[4]. Edge Impulse tutorial - Responding to your voice: https://docs.edgeimpulse.com/docs/tutorials/end-to-end-tutorials/responding-to-your-voice

[5]. Maqsood – Kalman Filters - 14_kalmanFilter.pdf