```
/* USER CODE BEGIN Header */
/**
******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
/* Private includes ----------------------------------------------------------*/
```

```
/* USER CODE BEGIN Includes */
#define VREF_ADC 3300
#define RefrencePositionLeftRight 2025
#define RefrencePositionUpDown 2100
#define Hys 10
#define On GPIO_PIN_RESET
#define Off GPIO_PIN_SET
#define Button_Click() HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13)
#define Button_Up() HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_10)
#define Button_Down() HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_5)
#define Button_Left() HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_4)
#define Button_Right() HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_2)
#define Blinker_On() HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, Off)
#define Blinker_Off() HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7, On)
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
void MeasureADCLR(float* value);
void MeasureADCUD(float* value);
void CentreGlassLR(float* valueLR);
void CentreGlassUD(float* valueUD);
void Move_UP();
void Move_DOWN();
void Move_LEFT();
void Move_RIGHT();
void DONT_MOVE();
/* USER CODE END PTD */
```

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_ADC2_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */
```

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */


/* USER CODE END 0 */


/**
  * @brief  The application entry point.
  * @retval int
  */
```

```
int main(void)
{
  /* USER CODE BEGIN 1 */
float value=0;
  /* USER CODE END 1 */
  /* MCU Configuration--------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_ADC2_Init();
  /* USER CODE BEGIN 2 */
 HAL_ADCEx_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);
 HAL_ADCEx_Calibration_Start(&hadc2, ADC_SINGLE_ENDED);
 CentreGlassLR(&value);
 CentreGlassUD(&value);
  /* USER CODE END 2 */
```

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
	//Blinker
	if (Button_Click() == On)
	{
		Blinker_On();
		HAL_Delay(200);
		Blinker_Off();
		HAL_Delay(200);
	}
	else
	{
		Blinker_Off();
	}
	//MirrorUD
	if (Button_Up() == On)
	{
		Move_UP();
	}
	if (Button_Down() == On)
	{
		Move_DOWN();
	}
	else if (Button_Up() == Off && Button_Down() == Off)
	{
		DONT_MOVE();
```

```
            }
            //MirrorLR
            if (Button_Left() == On)
            {
                  Move_LEFT();
            }
            if (Button_Right() == On)
            {
                  Move_RIGHT();
            }
            else if (Button_Left() == Off && Button_Right() == Off)
            {
                  DONT_MOVE();
            }
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
```

```
  /** Configure the main internal regulator output voltage
  */
  HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PC
LK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
```

```
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{
  /* USER CODE BEGIN ADC1_Init 0 */
  /* USER CODE END ADC1_Init 0 */
  ADC_MultiModeTypeDef multimode = {0};
  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */
  /* USER CODE END ADC1_Init 1 */
  /** Common config*/
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV16;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.GainCompensation = 0;
  hadc1.Init.ScanConvMode = ADC_SCAN_ENABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
```

```
hadc1.Init.LowPowerAutoWait = DISABLE;
hadc1.Init.ContinuousConvMode = DISABLE;
hadc1.Init.NbrOfConversion = 2;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
hadc1.Init.DMAContinuousRequests = DISABLE;
hadc1.Init.Overrun = ADC_OVR_DATA_PRESERVED;
hadc1.Init.OversamplingMode = DISABLE;
if (HAL_ADC_Init(&hadc1) != HAL_OK)
{
  Error_Handler();
}
/** Configure the ADC multi-mode
*/
multimode.Mode = ADC_MODE_INDEPENDENT;
if (HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode) != HAL_OK)
{
  Error_Handler();
}
/** Configure Regular Channel*/
sConfig.Channel = ADC_CHANNEL_7;
sConfig.Rank = ADC_REGULAR_RANK_1;
sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
sConfig.SingleDiff = ADC_SINGLE_ENDED;
sConfig.OffsetNumber = ADC_OFFSET_NONE;
sConfig.Offset = 0;
if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  /** Configure Regular Channel*/
  sConfig.Channel = ADC_CHANNEL_8;
  sConfig.Rank = ADC_REGULAR_RANK_2;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */
  /* USER CODE END ADC1_Init 2 */
}
/**
  * @brief ADC2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC2_Init(void)
{
  /* USER CODE BEGIN ADC2_Init 0 */
  /* USER CODE END ADC2_Init 0 */
  ADC_ChannelConfTypeDef sConfig = {0};
  /* USER CODE BEGIN ADC2_Init 1 */
  /* USER CODE END ADC2_Init 1 */
```

```
/** Common config*/
hadc2.Instance = ADC2;
hadc2.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV16;
hadc2.Init.Resolution = ADC_RESOLUTION_12B;
hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc2.Init.GainCompensation = 0;
hadc2.Init.ScanConvMode = ADC_SCAN_DISABLE;
hadc2.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
hadc2.Init.LowPowerAutoWait = DISABLE;
hadc2.Init.ContinuousConvMode = DISABLE;
hadc2.Init.NbrOfConversion = 1;
hadc2.Init.DiscontinuousConvMode = DISABLE;
hadc2.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc2.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
hadc2.Init.DMAContinuousRequests = DISABLE;
hadc2.Init.Overrun = ADC_OVR_DATA_PRESERVED;
hadc2.Init.OversamplingMode = DISABLE;
if (HAL_ADC_Init(&hadc2) != HAL_OK)
{
  Error_Handler();
}
/** Configure Regular Channel*/
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = ADC_REGULAR_RANK_1;
sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
sConfig.SingleDiff = ADC_SINGLE_ENDED;
sConfig.OffsetNumber = ADC_OFFSET_NONE;
sConfig.Offset = 0;
```

```
  if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }

  /* USER CODE BEGIN ADC2_Init 2 */
  /* USER CODE END ADC2_Init 2 */
}
/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */
/* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC,
GPIO_PIN_14|GPIO_PIN_15|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_10|GPIO_PIN_11,
GPIO_PIN_RESET);
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_15, GPIO_PIN_RESET);
```

```
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1|GPIO_PIN_5|GPIO_PIN_7|GPIO_PIN_9,
GPIO_PIN_RESET);
  /*Configure GPIO pins : PC13 PC4 PC5 */
  GPIO_InitStruct.Pin = GPIO_PIN_13|GPIO_PIN_4|GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
  /*Configure GPIO pins : PC14 PC15 PC7 PC8 PC10 PC11 */
  GPIO_InitStruct.Pin =
GPIO_PIN_14|GPIO_PIN_15|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_10|GPIO_PIN_11;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
  /*Configure GPIO pins : PA6 PA7 PA15 */
  GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  /*Configure GPIO pins : PB1 PB5 PB7 PB9 */
  GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_5|GPIO_PIN_7|GPIO_PIN_9;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  /*Configure GPIO pins : PB2 PB10 */
```

```
  GPIO_InitStruct.Pin = GPIO_PIN_2|GPIO_PIN_10;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  /*Configure GPIO pins : PB12 PB13 */
  GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
/* USER CODE BEGIN 4 */
void Move_UP(void)
{
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, Off);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, On);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, On);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, On);

        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_14, On);
        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_15, Off);
}
void Move_DOWN(void)
{
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, Off);
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, On);
        HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, On);
```

```
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, On);

	HAL_GPIO_WritePin(GPIOC, GPIO_PIN_14, On);
	HAL_GPIO_WritePin(GPIOC, GPIO_PIN_15, Off);
}
void Move_LEFT(void)
{
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, Off);
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, On);
	HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, On);
	HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, On);

	HAL_GPIO_WritePin(GPIOC, GPIO_PIN_14, Off);
	HAL_GPIO_WritePin(GPIOC, GPIO_PIN_15, On);
}
void Move_RIGHT(void)
{
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, Off);
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, On);
	HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, On);
	HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, On);

	HAL_GPIO_WritePin(GPIOC, GPIO_PIN_14, Off);
	HAL_GPIO_WritePin(GPIOC, GPIO_PIN_15, On);
}
void DONT_MOVE(void)
{
	HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, On);
```

```
	HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, On);
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, On);
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, On);
}
void CentreGlassLR(float* valueLR)
{
	MeasureADCLR(valueLR);
	while(*valueLR>(RefrencePositionLeftRight + Hys) ||
*valueLR<(RefrencePositionLeftRight - Hys))
	{
		if(*valueLR<(RefrencePositionLeftRight))
		{
			Move_RIGHT();
		}
		else
		{
			Move_LEFT();
		}
			MeasureADCLR(valueLR);
	}
}
void CentreGlassUD(float* valueUD)
{
	MeasureADCUD(valueUD);
	while(*valueUD>(RefrencePositionUpDown + Hys) ||
*valueUD<(RefrencePositionUpDown - Hys))
	{
		if(*valueUD<(RefrencePositionUpDown))
```

```
                {
                        Move_UP();
                }
                else
                {
                        Move_DOWN();
                }
                MeasureADCUD(valueUD);
        }
}
void MeasureADCLR(float* value)
{
        uint16_t valueraw=0;
        HAL_ADC_Start(&hadc1);
        while(HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY)!= HAL_OK)
        { }
        valueraw=HAL_ADC_GetValue(&hadc1);
        *value=__HAL_ADC_CALC_DATA_TO_VOLTAGE(VREF_ADC,valueraw,ADC_RESOLUTION_
12B);
}
void MeasureADCUD(float* value)
{
        uint16_t valueraw=0;
        HAL_ADC_Start(&hadc2);
        while(HAL_ADC_PollForConversion(&hadc2, HAL_MAX_DELAY)!= HAL_OK)
        { }
        valueraw=HAL_ADC_GetValue(&hadc2);
```

```
        *value=__HAL_ADC_CALC_DATA_TO_VOLTAGE(VREF_ADC,valueraw,ADC_RESOLUTION_
12B);
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
```

```
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```