```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stepmotor1.h"
#include "stepmotor2.h"
#include "stepmotor3.h"
#include "stepmotor4.h"
#include "stepmotor5.h"
#include "stepmotor6.h"
#include "servomotor7.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* 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 ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
volatile uint16_t figure = 0;

volatile uint16_t stepCounter1 = 0;
volatile uint16_t stepLimit1 = 0;
volatile uint16_t isStop1 = 1;
volatile int rotationCounter1 = 0;

volatile uint16_t stepCounter2 = 0;
volatile uint16_t stepLimit2 = 0;
```

```
63 volatile uint16_t isStop2 = 1;
64 volatile int rotationCounter2 = 0;
65
66 volatile uint16_t stepCounter3 = 0;
67 volatile uint16_t stepLimit3 = 0;
68 volatile uint16_t isStop3 = 1;
69 volatile int rotationCounter3 = 0;
70
71 volatile uint16_t stepCounter4 = 0;
72 volatile uint16_t stepLimit4 = 0;
73 volatile uint16_t isStop4 = 1;
74 volatile int rotationCounter4 = 0;
75
76 volatile uint16_t stepCounter5 = 0;
77 volatile uint16_t stepLimit5 = 0;
78 volatile uint16_t isStop5 = 1;
79 volatile int rotationCounter5 = 0;
80
81 volatile uint16_t stepCounter6 = 0;
82 volatile uint16_t stepLimit6 = 0;
83 volatile uint16_t isStop6 = 1;
84 volatile int rotationCounter6 = 0;
85 /* USER CODE END PV */
86
87 /* Private function prototypes -----------------------------------------------*/
88 void SystemClock_Config(void);
89 /* USER CODE BEGIN PFP */
90 void pick_move(void);
91 void circle_move(void);
92 void square_move(void);
93 void home_move(void);
94 void star_move(void);
95 void triangle_move(void);
96 /* USER CODE END PFP */
97
98 /* Private user code ---------------------------------------------------------*/
99 /* USER CODE BEGIN 0 */
100 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
101 {
102     if(htim == &htim8)
103     {
104         stepCounter1 = stepCounter1 + 1;
105         if(stepCounter1 >= stepLimit1)
106         {
107             HAL_TIM_PWM_Stop(&htim8, TIM_CHANNEL_2);
108             stepCounter1 = 0;
109             isStop1 = 1;
110         }
111     }
112     if(htim == &htim4)
113     {
114         stepCounter2 = stepCounter2 + 1;
115         if(stepCounter2 >= stepLimit2)
116         {
117             HAL_TIM_PWM_Stop(&htim4, TIM_CHANNEL_1);
118             stepCounter2 = 0;
119             isStop2 = 1;
120         }
121     }
122     if(htim == &htim14)
123     {
124         stepCounter3 = stepCounter3 + 1;
```

```
125         if(stepCounter3 >= stepLimit3)
126         {
127             HAL_TIM_PWM_Stop(&htim14, TIM_CHANNEL_1);
128             stepCounter3 = 0;
129             isStop3 = 1;
130         }
131     }
132 
133     if(htim == &htim1)
134     {
135         stepCounter4 = stepCounter4 + 1;
136         if(stepCounter4 >= stepLimit4)
137         {
138             HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_3);
139             stepCounter4 = 0;
140             isStop4 = 1;
141         }
142     }
143     if(htim == &htim2)
144     {
145         stepCounter5 = stepCounter5 + 1;
146         if(stepCounter5 >= stepLimit5)
147         {
148             HAL_TIM_PWM_Stop(&htim2, TIM_CHANNEL_2);
149             stepCounter5 = 0;
150             isStop5 = 1;
151         }
152     }
153     if(htim == &htim3)
154     {
155         stepCounter6 = stepCounter6 + 1;
156         if(stepCounter6 >= stepLimit6)
157         {
158             HAL_TIM_PWM_Stop(&htim3, TIM_CHANNEL_2);
159             stepCounter6 = 0;
160             isStop6 = 1;
161         }
162     }
163 }
164 /* USER CODE END 0 */
165 
166 /**
167   * @brief  The application entry point.
168   * @retval int
169   */
170 int main(void)
171 {
172   /* USER CODE BEGIN 1 */
173 
174   /* USER CODE END 1 */
175 
176   /* MCU Configuration--------------------------------------------------------*/
177 
178   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
179   HAL_Init();
180 
181   /* USER CODE BEGIN Init */
182 
183   /* USER CODE END Init */
184 
185   /* Configure the system clock */
186   SystemClock_Config();
```

```
187
188  /* USER CODE BEGIN SysInit */
189
190  /* USER CODE END SysInit */
191
192  /* Initialize all configured peripherals */
193  MX_GPIO_Init();
194  MX_USART2_UART_Init();
195  MX_TIM1_Init();
196  MX_TIM2_Init();
197  MX_TIM3_Init();
198  MX_TIM4_Init();
199  MX_TIM8_Init();
200  MX_TIM14_Init();
201  MX_TIM10_Init();
202  /* USER CODE BEGIN 2 */
203  ////////////////////
204  stepper_init_motor1();
205  stepper_init_motor2();
206  stepper_init_motor3();
207  stepper_init_motor4();
208  stepper_init_motor5();
209  stepper_init_motor6();
210  servo_init_motor7();
211  ////////////////////
212  //HAL_TIM_Base_Start_IT(&htim1);
213  //HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
214  /* USER CODE END 2 */
215
216  /* Infinite loop */
217  /* USER CODE BEGIN WHILE */
218  HAL_Delay(500);
219  servo_close_gripper();
220  HAL_Delay(3000);
221  while (1)
222  {
223      if(figure == 1)
224      {
225        pick_move();
226        square_move();
227        home_move();
228      }
229      if(figure == 2)
230      {
231        pick_move();
232        circle_move();
233        home_move();
234      }
235      if(figure == 3)
236      {
237        pick_move();
238        triangle_move();
239        home_move();
240      }
241      if(figure == 4)
242      {
243        pick_move();
244        star_move();
245        home_move();
246      }
247
248
```

```
249
250     /* USER CODE END WHILE */
251
252     /* USER CODE BEGIN 3 */
253   }
254   /* USER CODE END 3 */
255 }
256
257 /**
258   * @brief System Clock Configuration
259   * @retval None
260   */
261 void SystemClock_Config(void)
262 {
263   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
264   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
265
266   /** Configure the main internal regulator output voltage
267   */
268   __HAL_RCC_PWR_CLK_ENABLE();
269   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
270   /** Initializes the RCC Oscillators according to the specified parameters
271   * in the RCC_OscInitTypeDef structure.
272   */
273   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
274   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
275   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
276   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
277   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
278   RCC_OscInitStruct.PLL.PLLM = 8;
279   RCC_OscInitStruct.PLL.PLLN = 84;
280   RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
281   RCC_OscInitStruct.PLL.PLLQ = 2;
282   RCC_OscInitStruct.PLL.PLLR = 2;
283   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
284   {
285     Error_Handler();
286   }
287   /** Initializes the CPU, AHB and APB buses clocks
288   */
289   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
290                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
291   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
292   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
293   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
294   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
295
296   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
297   {
298     Error_Handler();
299   }
300 }
301
302 /* USER CODE BEGIN 4 */
303 /* Ruch robota w kierunku kwadratu */
304 void pick_move(void)
305 {
306     figure = 0;
307     HAL_Delay(2000);
308     servo_open_gripper();
309     stepper_rot_motor2(4000, 5, 0, &stepLimit2, &rotationCounter2, &isStop2);
310     stepper_rot_motor3(3500, 5, 0, &stepLimit3, &rotationCounter3, &isStop3);
```

```
311     stepper_rot_motor6(900, 1, 1, &stepLimit6, &rotationCounter6, &isStop6);
312     HAL_Delay(3000);
313     stepper_rot_motor5(250, 1, 1, &stepLimit5, &rotationCounter5, &isStop5);
314     while(isStop2 == 0);
315     while(isStop3 == 0);
316     while(isStop5 == 0);
317     while(isStop6 == 0);
318     HAL_Delay(500);
319     servo_close_gripper();
320 }
321
322 void square_move(void)
323 {
324     stepper_rot_motor2(2000, 5, 1, &stepLimit2, &rotationCounter2, &isStop2);
325     stepper_rot_motor3(1000, 5, 1, &stepLimit3, &rotationCounter3, &isStop3);
326     stepper_rot_motor5(125, 1, 0, &stepLimit5, &rotationCounter5, &isStop5);
327     while(isStop2 == 0);
328     while(isStop3 == 0);
329     while(isStop5 == 0);
330     stepper_rot_motor2(100, 5, 0, &stepLimit2, &rotationCounter2, &isStop2);
331     stepper_rot_motor5(80, 1, 1, &stepLimit5, &rotationCounter5, &isStop5);
332     while(isStop2 == 0);
333     stepper_rot_motor2(1500, 4, 0, &stepLimit2, &rotationCounter2, &isStop2);
334     stepper_rot_motor1(1350, 5, 0, &stepLimit1, &rotationCounter1, &isStop1);
335     stepper_rot_motor3(2000, 5, 0, &stepLimit3, &rotationCounter3, &isStop3);
336     while(isStop2 == 0);
337     while(isStop3 == 0);
338     while(isStop5 == 0);
339     while(isStop1 == 0);
340     HAL_Delay(500);
341     servo_open_gripper();
342 }
343
344 void circle_move(void)
345 {
346     stepper_rot_motor2(2000, 5, 1, &stepLimit2, &rotationCounter2, &isStop2);
347     stepper_rot_motor3(1000, 5, 1, &stepLimit3, &rotationCounter3, &isStop3);
348     stepper_rot_motor5(125, 1, 0, &stepLimit5, &rotationCounter5, &isStop5);
349     while(isStop2 == 0);
350     while(isStop3 == 0);
351     while(isStop5 == 0);
352     stepper_rot_motor2(100, 5, 0, &stepLimit2, &rotationCounter2, &isStop2);
353     stepper_rot_motor5(80, 1, 1, &stepLimit5, &rotationCounter5, &isStop5);
354     while(isStop2 == 0);
355     stepper_rot_motor2(1500, 4, 0, &stepLimit2, &rotationCounter2, &isStop2);
356     stepper_rot_motor1(1350, 5, 1, &stepLimit1, &rotationCounter1, &isStop1);
357     stepper_rot_motor3(2000, 5, 0, &stepLimit3, &rotationCounter3, &isStop3);
358     while(isStop2 == 0);
359     while(isStop3 == 0);
360     while(isStop5 == 0);
361     while(isStop1 == 0);
362     HAL_Delay(500);
363     servo_open_gripper();
364 }
365
366 void triangle_move(void)
367 {
368     stepper_rot_motor2(2500, 5, 1, &stepLimit2, &rotationCounter2, &isStop2);
369     stepper_rot_motor3(1500, 5, 1, &stepLimit3, &rotationCounter3, &isStop3);
370     stepper_rot_motor5(125, 1, 0, &stepLimit5, &rotationCounter5, &isStop5);
371     while(isStop2 == 0);
372     while(isStop3 == 0);
```

```
373     while(isStop5 == 0);
374     stepper_rot_motor2(100, 5, 0, &stepLimit2, &rotationCounter2, &isStop2);
375     stepper_rot_motor5(100, 1, 0, &stepLimit5, &rotationCounter5, &isStop5);
376     while(isStop2 == 0);
377     stepper_rot_motor1(4050, 6, 1, &stepLimit1, &rotationCounter1, &isStop1);
378     while(isStop5 == 0);
379     while(isStop1 == 0);
380     HAL_Delay(500);
381     servo_open_gripper();
382 }
383
384 void star_move(void)
385 {
386     stepper_rot_motor2(2400, 5, 1, &stepLimit2, &rotationCounter2, &isStop2);
387     stepper_rot_motor3(1400, 5, 1, &stepLimit3, &rotationCounter3, &isStop3);
388     stepper_rot_motor5(125, 1, 0, &stepLimit5, &rotationCounter5, &isStop5);
389     while(isStop2 == 0);
390     while(isStop3 == 0);
391     while(isStop5 == 0);
392     stepper_rot_motor2(100, 5, 0, &stepLimit2, &rotationCounter2, &isStop2);
393     stepper_rot_motor5(100, 1, 0, &stepLimit5, &rotationCounter5, &isStop5);
394     while(isStop2 == 0);
395     stepper_rot_motor1(4050, 6, 0, &stepLimit1, &rotationCounter1, &isStop1);
396     while(isStop5 == 0);
397     while(isStop1 == 0);
398     HAL_Delay(500);
399     servo_open_gripper();
400 }
401
402 void home_move(void)
403 {
404     HAL_Delay(500);
405     stepper_rot_home_motor2(5, 0, &stepLimit2, &rotationCounter2, &isStop2);
406     stepper_rot_home_motor3(5, 0, &stepLimit3, &rotationCounter3, &isStop3);
407     HAL_Delay(1000);
408     stepper_rot_home_motor1(5, 0, &stepLimit1, &rotationCounter1, &isStop1);
409     stepper_rot_home_motor5(1, 0, &stepLimit5, &rotationCounter5, &isStop5);
410     stepper_rot_home_motor6(1, 0, &stepLimit6, &rotationCounter6, &isStop6);
411     while(isStop1 == 0);
412     while(isStop2 == 0);
413     while(isStop3 == 0);
414     while(isStop5 == 0);
415     while(isStop6 == 0);
416     servo_close_gripper();
417     HAL_Delay(500);
418 }
419
420 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
421 {
422     if(GPIO_Pin == SensorSquare_Pin)
423     {
424         figure = 1;
425     }
426     if(GPIO_Pin == SensorRoller_Pin)
427     {
428         figure = 2;
429     }
430     if(GPIO_Pin == SensorTriangle_Pin)
431     {
432         figure = 3;
433     }
434     if(GPIO_Pin == SensorStar_Pin)
```

```
435     {
436         figure = 4;
437     }
438 }
439
440 /* USER CODE END 4 */
441
442 /**
443   * @brief  This function is executed in case of error occurrence.
444   * @retval None
445   */
446 void Error_Handler(void)
447 {
448   /* USER CODE BEGIN Error_Handler_Debug */
449   /* User can add his own implementation to report the HAL error return state */
450
451   /* USER CODE END Error_Handler_Debug */
452 }
453
454 #ifdef  USE_FULL_ASSERT
455 /**
456   * @brief  Reports the name of the source file and the source line number
457   *         where the assert_param error has occurred.
458   * @param  file: pointer to the source file name
459   * @param  line: assert_param error line source number
460   * @retval None
461   */
462 void assert_failed(uint8_t *file, uint32_t line)
463 {
464   /* USER CODE BEGIN 6 */
465   /* User can add his own implementation to report the file name and line number,
466      tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
467   /* USER CODE END 6 */
468 }
469 #endif /* USE_FULL_ASSERT */
470
471 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
472
```