```
/**
  ******************************************************************************
  * \mainpage
  *
  * \section
  *
  * Baby Robot:     Unité de tests pour le déplacement autonome d'un baby duck
  *                 dont la la tête se mouve de manière régulée dans une direction
  *                 donnée par le signal d'un potentiomètre. La position de la tête
  *                 permet la régulation des déplacements du robot. Le potentiomètre
  *                 de direction est intégré sur la tête du canard et donc se tourne
  *                 avec la tête du canard.
  *
  * @author   Frédéric Angéloz (BFH-TI, angef1)
  ******************************************************************************
  */

/**
  ******************************************************************************
  * @file     main.c
  * @brief    Baby Duck Robot
  * @author   BFH-TI / Angéloz Frédéric
  * @version  Final
  * @date     06.2014
  * @hardware Olimex H103
  ******************************************************************************
  */


/* Includes
------------------------------------------------------------------*/
#include "stm32f10x.h"
#include <stdio.h>
#include <math.h>

/* Private variables
---------------------------------------------------------------
*/
volatile extern u8 tickFlag;

/* Measure value for potentiometer */
  u16 poti;

/* PID values for head control */
  int Head_ticker=0;
  int Motors_ticker=0;
  float Head_PID;
  float Motors_PID;
  float dt;
  float command;
  float Poti_measure;
  float Head_measure;
  float error=0;
  float kp;
  float ki;
  float kd;
  float previous_error=0;
  float integral=0;
  float derivative=0;
  float output=0;
/* Command values for head moving */
  u16 Head_move;
  int delta;
  int k;
  int diff;
  u16 tmp3;

/* Command values for motors control */
  u16 PWMLenght=800;
  u16 Vmax;
  u16 Vmin;
  u16 VRight;
  u16 VLeft;
  u16 tmp1;
  u16 tmp2;
  int deltamot;

/* RC Variable for robot control */
```

```
  79     int speed;
  80     int direction;
  81     int duck;
  82     int start;
  83     int mode;
  84     u16 ch1, ch2, ch3, ch4, ch5, ch6;
  85     u16 fall1, fall2, fall3, fall4, fall5, fall6;
  86     u16 rise1, rise2, rise3, rise4, rise5, rise6;
  87     int state1=0, state2=0, state3=0, state4=0, state5=0, state6=0;
  88     int Duckcommand_X;
  89     int Duckcommand_Y;
  90     int StartSwitch;
  91     int ModeSwitch;
  92     int Speed_Joystick;
  93     int Direction_Joystick;
  94
  95   /* Private function prototypes
  96   -----------------------------------------------
  97   */
  98   void RCC_Configuration(void);
  99   void GPIO_Configuration(void);
 100   void NVIC_Configuration(void);
 101   void SYSTICK_Configuration(int SystemTickPeriod_us);
 102   void ADC1_Configuration(void);
 103   void Timer_Configuration(void);
 104   void PWMInit(void);
 105   float max(float num1, float num2);
 106   float min(float num1, float num2);
 107   void  TIM4_IRQHandler(void);
 108   void  TIM2_IRQHandler(void);
 109   void RC_Command(int *Start, int *Mode, int *Duck, int *Speed, int *Direction);
 110   float PID_Control(float kp, float ki, float kd, float dt, float command, float measure);
 111
 112   /* Private functions
 113   -------------------------------------------------------
 114   */
 115   ADC_InitTypeDef ADC_InitStructure;
 116   GPIO_InitTypeDef GPIO_InitStructure;
 117   NVIC_InitTypeDef NVIC_InitStructure;
 118   TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
 119   TIM_OCInitTypeDef TIM_OCInitStructure;
 120   TIM_ICInitTypeDef TIM_ICInitStructure;
 121   /* --- end of header ------------------------------------------------------ */
 122   /**
 123    ------------------------------------------------------------------------------
 124     * @name  main
 125     * @brief  Régulation boucle fermée du système. Recherche de la direction du
 126     *         signal par la tête du canard et déplacement du robot en fonction
 127     *         de la direction.
 128     * @param  None
 129     * @retval None
 130    ------------------------------------------------------------------------------
 131   */
 132   int main(void)
 133   {
 134     /* NVIC configuration */
 135     NVIC_Configuration();
 136     /* System Clocks Configuration */
 137     RCC_Configuration();
 138     /* Configure the GPIO ports */
 139     GPIO_Configuration();
 140     /* Configure the system ticker to 1ms */
 141     SYSTICK_Configuration(1000);
 142     /*  Configure the ADC1*/
 143     ADC1_Configuration();
 144     /*  Configure the TIM1*/
 145     Timer_Configuration();
 146     /* Speed values */
 147     Vmin=PWMLenght/10;
 148     Vmax=PWMLenght;
 149
 150     while(1)
 151     { /* Start of program
 152       -------------------------------*/
 153       RC_Command(&start, &mode, &duck, &speed, &direction);
 154
 155       /* Switch case for start or stop*/
 156       switch(start){
```

```
157         /* Stop */
158         case 0:
159           /* Default values if stop selected */
160           Head_move = 75;
161           VLeft = VRight = 0;
162           PWMInit();
163           break;
164 
165         /* Start */
166         case 1:
167 
168         /* Duck selection */
169         switch(duck){
170           /* Settings values if no duck is selected */
171           case 0:
172             VLeft = VRight = 0;
173             PWMInit();
174             break;
175 
176           /* duck_1 */
177           case 1:
178 
179             /* Mode selection */
180             switch(mode){
181 
182               /* Free mode */
183               case 1:
184                 /* Head orientation in free mode */
185                 Head_move = 75;                     //Middle position
186 
187                 /* If condition for direction */
188                 if(direction==0){                   //Straight ahead
189                 VLeft = VRight = Vmin*speed;
190                 }else {
191                     VLeft = (Vmin*speed)/fabs(min(direction, 1));
192                     VRight = (Vmin*speed)/fabs(max(direction, -1));
193                 }
194                 PWMInit();
195               break;
196 
197             /* Home mode */
198             case 2:
199               /* Potentiometer value for 180° of detection */
200                 poti = ADC_GetConversionValue(ADC1);
201 
202                 /* Head control
203                 ----------------------------------------------------------------*/
204 
205                 /* Potentiometre measure and conversion */
206                 Poti_measure = ((poti-1700.0)-400.0)/400.0;
207                 Poti_measure = max(Poti_measure, -1.0);
208                 Poti_measure = min(Poti_measure, 1.0);
209 
210                 /* Ticker for head regulation: 1ms (systick) */
211                 if(tickFlag){
212                   Head_ticker++;                    // Count up PID_ticker
213                   tickFlag = 0;                     // Reset tickFlag
214                 }
215                 /* PID Controller */
216                 if(Head_ticker==1){
217                   PID_Control(0.9,0.08,0.0,0.001,0.0,Poti_measure);
218                   Head_PID=output;
219                   Head_ticker=0;                    //Reset PID_ticker
220                 }
221                 /* Calcul for head orientation */
222                 delta=(Head_PID*1000)+1000;
223                 k=30;                               //minimal value for 0°
224                 diff=90;                            //difference between 0° value and 180° value
225                 Head_move=k+(delta/(2000/diff));    //algorithm for head rotation
226 
227               /* Motors control
228               ----------------------------------------------------------------*/
229 
230               /* Measure for closed loop */
231               Head_measure= ((Head_move-30.0)-45.0)/45.0;
232 
233               /* Ticker for motors regulation: 1ms (systick) */
234               if(tickFlag){
```

```
235                 Motors_ticker++;                    // Count up PID_ticker
236                 tickFlag = 0;                       // Reset tickFlag
237               }
238
239               /* PID Controller */
240               if(Motors_ticker==1){
241                 PID_Control(0.9,0.08,0.0,0.001,0.0,Head_measure);
242                 Motors_PID=output;
243                 Motors_ticker=0;                    //Reset PID_ticker
244
245
246                 if(Motors_PID==0.0){                //Straight ahead
247                 VLeft = VRight = Vmax;
248                 }else {
249                     VLeft = Vmax-fabs(min(Motors_PID, 0.0))*Vmax;
250                     VRight = Vmax-fabs(max(Motors_PID, 0.0))*Vmax;
251                 }
252               }
253
254               PWMInit();
255               break;
256             }
257             break;
258
259           /* Other ducks control (empty) */
260           case 2:
261             break;
262           case 3:
263             break;
264           case 4:
265             break;
266         }
267         break;
268       }
269     }
270   }
271
272   /**
273   ------------------------------------------------------------------------------
274   * Function Name : RC_Command
275   * Description   : Put the RC commands on pointers
276   * Input         : *Start, *Switch, *Duck, *Speed, *Direction
277   * Output         : None
278   * Return         : None
279   ------------------------------------------------------------------------------
280   */
281   void RC_Command(int *Start, int *Mode, int *Duck, int *Speed, int *Direction){
282     /* Start Switch */
283     if(StartSwitch>=1800 && StartSwitch<=2000){
284       *Start=1;
285     }else if(StartSwitch<=1180 && StartSwitch>=1000){
286       *Start=0;
287     }
288     /* Mode switch: free or home */
289     if(ModeSwitch>=1800 && ModeSwitch<=2000){
290       *Mode=1;
291     }else if(ModeSwitch<=1180 && ModeSwitch>=1000){
292       *Mode=2;
293     }
294     /* Duck choice */
295     if(Duckcommand_Y>=1800 && Duckcommand_Y<=2000){
296       *Duck=1;
297     }else if(Duckcommand_Y<=1180 && Duckcommand_Y>=1000){
298       *Duck=2;
299     }else if(Duckcommand_X>=1800 && Duckcommand_X<=2000){
300       *Duck=3;
301     }else if(Duckcommand_X<=1180 && Duckcommand_X>=1000){
302       *Duck=4;
303     }else{
304       *Duck=0;
305     }
306     /* Speed factor */
307     *Speed = (Speed_Joystick-1080)/82;
308     *Speed = max(*Speed,0);
309     *Speed = min(*Speed,10);
310
311     /* Direction value */
312     *Direction = ((Direction_Joystick-1080)-410)/41;
```

```
313     *Direction = max(*Direction, (-10));
314     *Direction = min(*Direction, 10);
315   }
316   /**
317   ------------------------------------------------------------------------------
318   * Function Name : PID_Control
319   * Description   : PID closed loop
320   * Input         : kp, ki, kd, dt, command, measure
321   * Output        : output
322   * Return        : None
323   ------------------------------------------------------------------------------
324   */
325   float PID_Control(float kp, float ki, float kd, float dt, float command, float measure){
326     error = command - measure;
327     integral = integral + error*dt;
328     derivative = (error - previous_error)/dt;
329     output = kp*error + ki*integral + kd*derivative;
330     previous_error = error;
331
332     output = max(output, -1.0);
333     output = min(output, 1.0);
334
335     integral = max(integral, -1.0);
336     integral = min(integral, 1.0);
337
338     derivative = max(integral, -1.0);
339     derivative = min(integral, 1.0);
340
341     return output;
342   }
343   /**
344   ------------------------------------------------------------------------------
345   * Function Name : PWMInit
346   * Description   : PWM initialisation if change occurs
347   * Input         : None
348   * Output        : None
349   * Return        : None
350   ------------------------------------------------------------------------------
351   */
352   void PWMInit(void){
353     if(tmp1 != VRight){
354       tmp1=VRight;
355       TIM_OCInitStructure.TIM_Pulse = VRight;
356       TIM_OC1Init(TIM1, &TIM_OCInitStructure);
357     }
358     if(tmp2 != VLeft){
359       tmp2=VLeft;
360       TIM_OCInitStructure.TIM_Pulse = VLeft;
361       TIM_OC2Init(TIM1, &TIM_OCInitStructure);
362     }
363     if(tmp3 != Head_move){
364       tmp3=Head_move;
365       TIM_OCInitStructure.TIM_Pulse = Head_move;
366       TIM_OC1Init(TIM3, &TIM_OCInitStructure);
367     }
368   }
369
370   /**
371   ------------------------------------------------------------------------------
372   * Function Name : max
373   * Description   : Return the max value between two
374   * Input         : num1, num2
375   * Output        : None
376   * Return        : result
377   ------------------------------------------------------------------------------
378   */
379   float max(float num1, float num2){
380      float result;
381      if (num1 > num2)
382         result = num1;
383      else
384         result = num2;
385      return result;
386   }
387   /**
388   ------------------------------------------------------------------------------
389   * Function Name : min
390   * Description   : Return the min value between two
```

```
391   * Input         : num1, num2
392   * Output        : None
393   * Return        : result
394   -------------------------------------------------------------------------------
395   */
396   float min(float num1, float num2){
397      float result;
398      if (num1 < num2)
399         result = num1;
400      else
401         result = num2;
402      return result;
403   }
404
405   /**
406   -------------------------------------------------------------------------------
407   * Function Name : ADC1_Configuration
408   * Description   : Configures the ADC1 for poti
409   * Input         : None
410   * Output        : None
411   * Return        : None
412   -------------------------------------------------------------------------------
413   */
414   void ADC1_Configuration(void)
415   {
416     /* Put everything back to power-on defaults */
417     ADC_DeInit(ADC1);
418     /* ADC1 Configuration
419     ----------------------------------------------------
420     */
421     /* ADC1, ADC2 operate independently */
422     ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
423     /* Disable the scan conversion so we do one at a time */
424     ADC_InitStructure.ADC_ScanConvMode = ENABLE;
425     /* Don't do continuous conversions - do them on demand */
426     ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
427     /* Start conversin by software, not an external trigger */
428     ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
429     /* Conversions are 12 bit - put them in the lower 12 bits of the result */
430     ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
431     /* Say how many channels would be used by the sequencer */
432     ADC_InitStructure.ADC_NbrOfChannel = 1;
433
434     /* Now do the setup */
435     ADC_Init(ADC1, &ADC_InitStructure);
436     /* ADC1 regular channel2 configuration */
437     ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_239Cycles5);
438
439     /* Enable ADC1 */
440     ADC_Cmd(ADC1, ENABLE);
441
442     /* Enable ADC1 reset calibaration register */
443     ADC_ResetCalibration(ADC1);
444     /* Check the end of ADC1 reset calibration register */
445     while(ADC_GetResetCalibrationStatus(ADC1));
446     /* Start ADC1 calibaration */
447     ADC_StartCalibration(ADC1);
448     /* Check the end of ADC1 calibration */
449     while(ADC_GetCalibrationStatus(ADC1));
450
451     /* Start ADC1 Software Conversion */
452     ADC_SoftwareStartConvCmd(ADC1, ENABLE);
453   }
454   /**
455   -------------------------------------------------------------------------------
456   * Function Name : Timer_Configuration
457   * Description   : Configures the TIM3 & the PWM Output
458   * Input         : None
459   * Output        : None
460   * Return        : None
461   -------------------------------------------------------------------------------
462   */
463   void Timer_Configuration(){
464
465     /* Motors control timer
466     ----------------------------------------------------------------*/
467
468     /* Time Base configuration for TIM1: Motors PWM control */
```

```
469     TIM_TimeBaseStructure.TIM_Prescaler = (72/(PWMLenght/400))-1;
470     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
471     TIM_TimeBaseStructure.TIM_Period = PWMLenght-1;     //Maximal value for pulse width
472     TIM_TimeBaseStructure.TIM_ClockDivision = 0;
473     TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;
474     TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);
475
476     /* Channel 1 in PWM Output mode for right motor */
477     TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;
478     TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
479     TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
480     TIM_OCInitStructure.TIM_Pulse =VRight;              // Duty cycle value
481     TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
482     TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
483     TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;
484     TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Reset;
485     TIM_OC1Init(TIM1, &TIM_OCInitStructure);
486
487       /* Channel 2 in PWM Output mode for left motor */
488     TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;
489     TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
490     TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
491     TIM_OCInitStructure.TIM_Pulse = VLeft;              // Duty cycle value
492     TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
493     TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
494     TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;
495     TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Reset;
496     TIM_OC2Init(TIM1, &TIM_OCInitStructure);
497
498     TIM_Cmd(TIM1, ENABLE);
499     TIM_CtrlPWMOutputs(TIM1, ENABLE);
500
501     /* RC command inputs timer: Switches
502     ------------------------------------------------------------*/
503
504     /* Time base configuration for TIM2: RC inputs Start/Stop, Free/Home*/
505     TIM_TimeBaseStructure.TIM_Prescaler = 71;
506     TIM_TimeBaseStructure.TIM_Period = 0xFFFF;
507     TIM_TimeBaseStructure.TIM_ClockDivision = 0;
508     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
509     TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
510
511     TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
512     TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
513     TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
514     TIM_ICInitStructure.TIM_ICFilter = 0x0;
515     /* Channel 3 in PWM input mode for free/home mode switch */
516     TIM_ICInitStructure.TIM_Channel = TIM_Channel_3;
517     TIM_PWMIConfig(TIM2, &TIM_ICInitStructure);
518     /* Channel 4 in PWM input mode for start/stop switch */
519     TIM_ICInitStructure.TIM_Channel = TIM_Channel_4;
520     TIM_PWMIConfig(TIM2, &TIM_ICInitStructure);
521
522     /* Select the slave Mode: Reset Mode */
523     TIM_SelectSlaveMode(TIM2, TIM_SlaveMode_Reset);
524     TIM_SelectMasterSlaveMode(TIM2,TIM_MasterSlaveMode_Enable);
525
526     /* TIM enable counter */
527     TIM_Cmd(TIM2, ENABLE);
528
529     /* Enable the CC3, CC4 Interrupt Request */
530     TIM_ITConfig(TIM2, TIM_IT_CC3, ENABLE);
531     TIM_ITConfig(TIM2, TIM_IT_CC4, ENABLE);
532
533     /* Head control timer
534     ------------------------------------------------------------*/
535
536     /* Time Base configuration for TIM3: head control */
537     TIM_TimeBaseStructure.TIM_Prescaler = 1440-1;
538     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
539     TIM_TimeBaseStructure.TIM_Period = 2000-1;
540     TIM_TimeBaseStructure.TIM_ClockDivision = 0;
541     TIM_TimeBaseStructure.TIM_RepetitionCounter = 0;
542     TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
543
544       /* Channel 1 in PWM Output mode */
545     TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM2;
546     TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
```

```
547     TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Enable;
548     TIM_OCInitStructure.TIM_Pulse = Head_move;
549     TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
550     TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_High;
551     TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;
552     TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Reset;
553     TIM_OC1Init(TIM3, &TIM_OCInitStructure);
554
555     TIM_Cmd(TIM3, ENABLE);
556     TIM_CtrlPWMOutputs(TIM3, ENABLE);
557
558     /* RC command inputs timer: Joysticks
559     ----------------------------------------------------------------*/
560
561     /* Time base configuration for TIM4: Joystick */
562     TIM_TimeBaseStructure.TIM_Prescaler = 71;
563     TIM_TimeBaseStructure.TIM_Period = 0xFFFF;
564     TIM_TimeBaseStructure.TIM_ClockDivision = 0;
565     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
566     TIM_TimeBaseInit(TIM4, &TIM_TimeBaseStructure);
567
568     TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
569     TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
570     TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
571     TIM_ICInitStructure.TIM_ICFilter = 0x0;
572
573     /* Channel 1 in PWM input mode for direction joystick */
574     TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
575     TIM_PWMIConfig(TIM4, &TIM_ICInitStructure);
576     /* Channel 2 in PWM input mode for y-axis duck choice*/
577     TIM_ICInitStructure.TIM_Channel = TIM_Channel_2;
578     TIM_PWMIConfig(TIM4, &TIM_ICInitStructure);
579     /* Channel 3 in PWM input mode for speed joystick */
580     TIM_ICInitStructure.TIM_Channel = TIM_Channel_3;
581     TIM_PWMIConfig(TIM4, &TIM_ICInitStructure);
582     /* Channel 4 in PWM input mode for x-axis duck choice */
583     TIM_ICInitStructure.TIM_Channel = TIM_Channel_4;
584     TIM_PWMIConfig(TIM4, &TIM_ICInitStructure);
585
586     /* Select the slave Mode: Reset Mode */
587     TIM_SelectSlaveMode(TIM4, TIM_SlaveMode_Reset);
588     TIM_SelectMasterSlaveMode(TIM4,TIM_MasterSlaveMode_Enable);
589
590     /* TIM enable counter */
591     TIM_Cmd(TIM4, ENABLE);
592
593     /* Enable the CC2 Interrupt Request */
594     TIM_ITConfig(TIM4, TIM_IT_CC1, ENABLE);
595     TIM_ITConfig(TIM4, TIM_IT_CC2, ENABLE);
596     TIM_ITConfig(TIM4, TIM_IT_CC3, ENABLE);
597     TIM_ITConfig(TIM4, TIM_IT_CC4, ENABLE);
598   }
599
600   /*******************************************************************************
601   * Function Name : RCC_Configuration
602   * Description   : Configures the different system clocks.
603   * Input         : None
604   * Output         : None
605   * Return         : None
606   *******************************************************************************/
607   void RCC_Configuration(void)
608   {
609     /* Enable clock for TIM1, GPIOA, GPIOB, GPIOC, AFIO and ADC1 */
610     RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1 |
611                            RCC_APB2Periph_GPIOA |
612                            RCC_APB2Periph_GPIOC |
613                            RCC_APB2Periph_GPIOB |
614                            RCC_APB2Periph_AFIO |
615                            RCC_APB2Periph_ADC1, ENABLE);
616
617     /* Enable clock for TIM2, TIM3 and TIM4 */
618     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2 |
619                            RCC_APB1Periph_TIM3 |
620                            RCC_APB1Periph_TIM4, ENABLE);
621
622   }
623
624   /*******************************************************************************
```

```
625   * Function Name : GPIO_Configuration
626   * Description   : Configures the different GPIO ports.
627   * Input         : None
628   * Output         : None
629   * Return         : None
630   *******************************************************************************/
631   void GPIO_Configuration(void)
632   {
633     GPIO_InitTypeDef GPIO_InitStructure;
634     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
635
636     /* Configure PC12 as Output fo LED */
637     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
638     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
639     GPIO_Init(GPIOC, &GPIO_InitStructure);
640
641     /* TIM1 Channel 1, 2 as alternate function for DC motors PWM */
642     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;
643     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
644     GPIO_Init(GPIOA, &GPIO_InitStructure);
645
646       /* TIM3 Channel 1 as alternate function for head PWM */
647     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
648     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
649     GPIO_Init(GPIOA, &GPIO_InitStructure);
650
651       /* Configure PA1 as analog input for potentiometer */
652     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
653     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
654     GPIO_Init(GPIOA, &GPIO_InitStructure);
655
656       /* TIM2 Channel 3, 4 as input floating for switches on RC command */
657     GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_2 | GPIO_Pin_3;
658     GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IN_FLOATING;
659     GPIO_Init(GPIOA, &GPIO_InitStructure);
660
661     /* TIM4 Channel 1, 2, 3, 4  as input floating for RC command */
662     GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_6 | GPIO_Pin_7 | GPIO_Pin_8| GPIO_Pin_9 ;
663     GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IN_FLOATING;
664     GPIO_Init(GPIOB, &GPIO_InitStructure);
665   }
666
667   /*******************************************************************************
668   * Function Name : NVIC_Configuration
669   * Description   : Configures Vector Table base location.
670   * Input         : None
671   * Output         : None
672   * Return         : None
673   *******************************************************************************/
674   void NVIC_Configuration(void)
675   {
676     NVIC_InitTypeDef NVIC_InitStructure;
677
678     #ifdef  VECT_TAB_RAM
679       /* Set the Vector Table base location at 0x20000000 */
680       NVIC_SetVectorTable(NVIC_VectTab_RAM, 0x0);
681     #else /* VECT_TAB_FLASH */
682       /* Set the Vector Table base location at 0x08000000 */
683       NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
684     #endif
685
686     /* Enable the TIM1 global Interrupt */
687     NVIC_InitStructure.NVIC_IRQChannel = TIM1_CC_IRQn;
688     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
689     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
690     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
691     NVIC_Init(&NVIC_InitStructure);
692
693     /* Enable the TIM2 global Interrupt */
694     NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
695     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
696     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
697     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
698     NVIC_Init(&NVIC_InitStructure);
699
700     /* Enable the TIM3 global Interrupt */
701     NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
702     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
```

```
703     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
704     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
705     NVIC_Init(&NVIC_InitStructure);
706
707     /* Enable the TIM4 global Interrupt */
708     NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
709     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
710     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
711     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
712     NVIC_Init(&NVIC_InitStructure);
713
714     NVIC_SetPriority(SysTick_IRQn,1);
715     NVIC_SetPriority(TIM3_IRQn,2);
716     NVIC_SetPriority(TIM1_CC_IRQn,3);
717
718   }
719   /*******************************************************************************
720   * Function Name : TIM2_IRQHandler
721   * Description   : Interrupt Request for timer 2
722   * Input         : None
723   * Output         : None
724   * Return         : None
725   *******************************************************************************/
726   void TIM2_IRQHandler(void)
727   {
728     if (TIM_GetITStatus(TIM2, TIM_IT_CC3) != RESET) {
729       TIM_ClearITPendingBit(TIM2, TIM_IT_CC3);
730       ch5 = TIM_GetCapture3(TIM2);
731               if (state5 == 0) {
732               rise5 = ch5;
733               TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Falling;
734               TIM_ICInitStructure.TIM_Channel = TIM_Channel_3;
735               TIM_ICInit(TIM2, &TIM_ICInitStructure);
736               state5 = 1;
737           } else {
738               fall5 = ch5;
739               ModeSwitch = fall5 - rise5;
740               TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
741               TIM_ICInitStructure.TIM_Channel = TIM_Channel_3;
742               TIM_ICInit(TIM2, &TIM_ICInitStructure);
743               state5 = 0;
744           }
745     }
746     if (TIM_GetITStatus(TIM2, TIM_IT_CC4) != RESET)   {
747         TIM_ClearITPendingBit(TIM2, TIM_IT_CC4);
748         ch6 = TIM_GetCapture4(TIM2);
749               if (state6 == 0) {
750               rise6 = ch6;
751               TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Falling;
752               TIM_ICInitStructure.TIM_Channel = TIM_Channel_4;
753               TIM_ICInit(TIM2, &TIM_ICInitStructure);
754               state6 = 1;
755           } else {
756               fall6 = ch6;
757               StartSwitch= fall6 - rise6;
758               TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
759               TIM_ICInitStructure.TIM_Channel = TIM_Channel_4;
760               TIM_ICInit(TIM2, &TIM_ICInitStructure);
761               state6 = 0;
762           }
763       }
764
765   }
766   /*******************************************************************************
767   * Function Name : TIM4_IRQHandler
768   * Description   : Interrupt Request for timer 4
769   * Input         : None
770   * Output         : None
771   * Return         : None
772   *******************************************************************************/
773   void TIM4_IRQHandler(void)
774   {
775     if (TIM_GetITStatus(TIM4, TIM_IT_CC1) != RESET) {
776       TIM_ClearITPendingBit(TIM4, TIM_IT_CC1);
777       ch1 = TIM_GetCapture1(TIM4);
778               if (state1 == 0) {
779               rise1 = ch1;
780               TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Falling;
```

```
781                 TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
782                 TIM_ICInit(TIM4, &TIM_ICInitStructure);
783                 state1 = 1;
784             } else {
785                 fall1 = ch1;
786                 Direction_Joystick = fall1 - rise1;
787                 TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
788                 TIM_ICInitStructure.TIM_Channel = TIM_Channel_1;
789                 TIM_ICInit(TIM4, &TIM_ICInitStructure);
790                 state1 = 0;
791             }
792         }
793         if (TIM_GetITStatus(TIM4, TIM_IT_CC2) != RESET)   {
794             TIM_ClearITPendingBit(TIM4, TIM_IT_CC2);
795             ch2 = TIM_GetCapture2(TIM4);
796                   if (state2 == 0) {
797                   rise2 = ch2;
798                   TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Falling;
799                   TIM_ICInitStructure.TIM_Channel = TIM_Channel_2;
800                   TIM_ICInit(TIM4, &TIM_ICInitStructure);
801                   state2 = 1;
802             } else {
803                   fall2 = ch2;
804                   Duckcommand_Y= fall2 - rise2;
805                   TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
806                   TIM_ICInitStructure.TIM_Channel = TIM_Channel_2;
807                   TIM_ICInit(TIM4, &TIM_ICInitStructure);
808                   state2 = 0;
809             }
810           }
811             if (TIM_GetITStatus(TIM4, TIM_IT_CC3) != RESET)   {
812             TIM_ClearITPendingBit(TIM4, TIM_IT_CC3);
813             ch3 = TIM_GetCapture3(TIM4);
814                if (state3 == 0) {
815                   rise3 = ch3;
816                   TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Falling;
817                   TIM_ICInitStructure.TIM_Channel = TIM_Channel_3;
818                   TIM_ICInit(TIM4, &TIM_ICInitStructure);
819                   state3 = 1;
820             } else {
821                   fall3 = ch3;
822                   Speed_Joystick = fall3 - rise3;
823                   TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
824                   TIM_ICInitStructure.TIM_Channel = TIM_Channel_3;
825                   TIM_ICInit(TIM4, &TIM_ICInitStructure);
826                   state3 = 0;
827             }
828           }
829         if (TIM_GetITStatus(TIM4, TIM_IT_CC4) != RESET)   {
830             TIM_ClearITPendingBit(TIM4, TIM_IT_CC4);
831             ch4 = TIM_GetCapture4(TIM4);
832                 if (state4 == 0) {
833                   rise4 = ch4;
834                   TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Falling;
835                   TIM_ICInitStructure.TIM_Channel = TIM_Channel_4;
836                   TIM_ICInit(TIM4, &TIM_ICInitStructure);
837                   state4 = 1;
838             } else {
839                   fall4 = ch4;
840                   Duckcommand_X = fall4 - rise4;
841                   TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
842                   TIM_ICInitStructure.TIM_Channel = TIM_Channel_4;
843                   TIM_ICInit(TIM4, &TIM_ICInitStructure);
844                   state4 = 0;
845             }
846           }
847
848   }
849
850
851   /*******************************************************************************
852   * Function Name : SYSTICK_Configuration
853   * Description   : Sets the Period of the system ticker
854   * Input         : Periode in us
855   * Output         : None
856   * Return         : None
857   *******************************************************************************/
858   void SYSTICK_Configuration(int SystemTickPeriod_us)
```

```
859   {
860     if (SysTick_Config(SystemCoreClock / (1000000/SystemTickPeriod_us) )) /* SystemClock is
861     defined in “system_stm32f10x.h” and equal to HCLK frequency */
862     {
863       /* Capture error */
864       while (1) {}
865     }
866   }
867
868   /******END OF FILE****/
869
```