Senior Seminar Project:

Robotic Pong Player

Braden Elliott

Table of Contents

Introduction 4

Statement of Problem 4

  Client Specifications 4

Project Design 4

  Electrical Design 4

  Mechanical Design 10

  Software Design 11

  Economic Constraints 12

Results 13

Implications for Future Work 13

References 14

Appendices 15

  Appendix A: Arm Controls Header and Source Code Files 15

  Appendix B: Main Source Code File for Arm Movement Demo 27

  Appendix C: Image Recognition Code File 29

Table of Figures

Figure 1: PCB Parts List 5

Figure 2: Circuit Schematic 7

Figure 3: PCB Board Layout 8

Figure 4: PCB 9

Figure 5: Arm Modeling 10

Figure 6: Robotic Arm 11

Introduction

A fun, simple game such a beer pong played between two humans is a great time. Bringing a robot into the mix could be an even better one. One arm, one eye, and a brain are needed to play, so make a robotic arm to throw ping pong balls, a camera to see where the ping pong balls go, and program a computer to decide what to do next in the game to win against the human.

Statement of Problem

The project is to design and build a robotic arm to be able to pick up and throw a ping pong ball and program a computer to interface with the arm and a camera to recognize where the ball was through relative to the cup layout and tell the arm what to do next. The visual feedback should tell whether the ball landed in a cup along with the specific cup it landed in.

Client Specifications

The client requires a proven concept of a robotic machine capable of performing the necessary tasks to play a human in a physical game such as pong. There should be a multi-jointed arm moving with multiple axis, image processing feedback, and external communication capability.

Project Design

Electrical Design

The arm is controlled via a custom motherboard which includes; one STM32F446RE microcontroller (Semiconductor Manufacturing, 2018) for I/O, two dual-H-bridges for dc motors, several pull-up circuits for servo motors, USB port for computer communication, three buck converters for power, and a transformer/rectifier circuit for wall power capability. The full list of components is shown in *Table 1*.

Part	Value	Device	Description
B1		SKB	SEMIKRON RECTIFIER
C1	100n	C-USC0603K	CAPACITOR
C2	100n	C-USC0603K	CAPACITOR
C3	100n	C-USC0603K	CAPACITOR
C4	100n	C-USC0603K	CAPACITOR
C5	100n	C-USC0603K	CAPACITOR
C6	100n	C-USC0603K	CAPACITOR
C7	100n	C-USC0603K	CAPACITOR
C8	4.7u	C-USC0603K	CAPACITOR
C9	100n	C-USC0603K	CAPACITOR
C10	1u	C-US150-054X183	CAPACITOR
C11	1u	C-US150-054X183	CAPACITOR
C12	1u	C-USC0603K	CAPACITOR
JP1		PINHD-2X10	PIN HEADER
LED1		LED_E	LED
R1	10k	R-US_MELF0204W	RESISTOR
R2	510	R-US_MELF0204W	RESISTOR
R3	100k	R-US_MELF0204W	RESISTOR
S1		SKHMPSE010	6.2 X 6.5mm TACT Switch (SMD)
SERVO_SUPPLY		PINHD-2X5	PIN HEADER
ST-LINK		PINHD-1X4	PIN HEADER
U$1		SN754410NE	Quadruple Half-H Driver
U$2		MOTOR_241F8X	Motor with an encoder
U$3		MOTOR_241F8X	Motor with an encoder
U$4		SN754410NE	Quadruple Half-H Driver
U$5		MOTOR_241F8X	Motor with an encoder
U$6		STM32F446RE	
U$7		POLOLU_BUCK	
U$8		POLOLU_BUCK	
U$9		POLOLU_BUCK	
U$11		+/-	
U$12		TRANSFORMER_PSS5-12	Transformer
U$13		+/-	
X1		MINI-USB_SHIELD5P4-32005-400	MINI USB-B R/A W/O POST 5 pol.

Figure 1: PCB Parts List

The schematic and PCB were designed in EAGLE, then ordered from Advanced PCB. It was put together and tested in KC138 at the University of Evansville. The schematic and PCB are shown in *Figures 2, 3*, and *4*. I wanted the power supply to be lenient for user satisfaction, so it was designed to be powered via a battery between seven and twenty-four volts or through the wall. If it plugged into the wall the power goes through a twelve-volt transformer and rectifier circuit to power the rest of the system. This circuit is bypassed if connected to a battery. The power supply voltages needed for each circuit module are either three, five, and/or six volts. A buck converter is used to drop the voltage down to each of those voltages. The controller is powered via three volts and several capacitors between the supply and ground pins. There is a reset circuit for holding the reset pin low until steady state power is reached along with a button for easy reset. There are five external pull-up circuits for the servo motors with header pins to connect the board to the motors on the arm. The fingers use three bidirectional motors powered by h-bridges and controlled via the magnetic encoders on the back of each motor. For PC communication, a USB circuit is used connected to a type-B port. The resistor R4 is for the host to determine Full Speed configuration. R3 pulls down the address pin since only one of these devices is used at a time.

Figure 2: Circuit Schematic

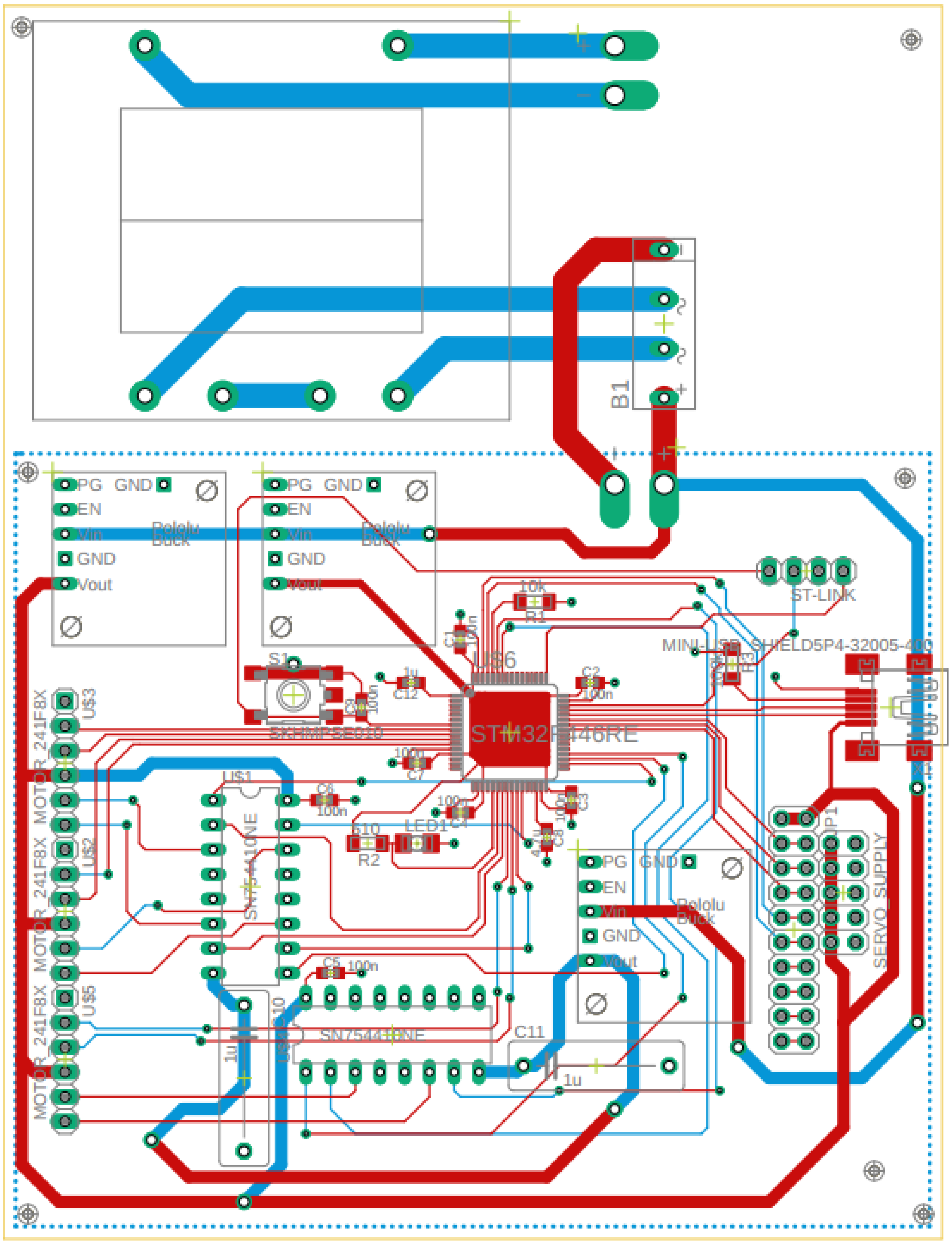
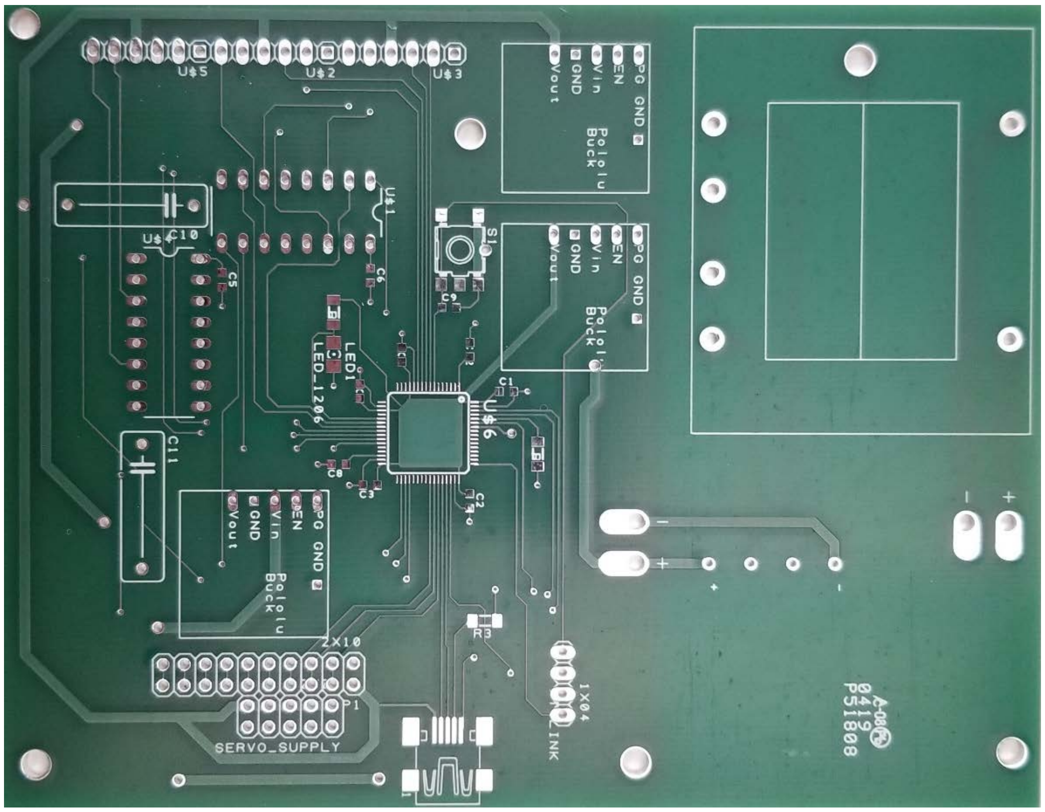


Figure 3: PCB Board Layout



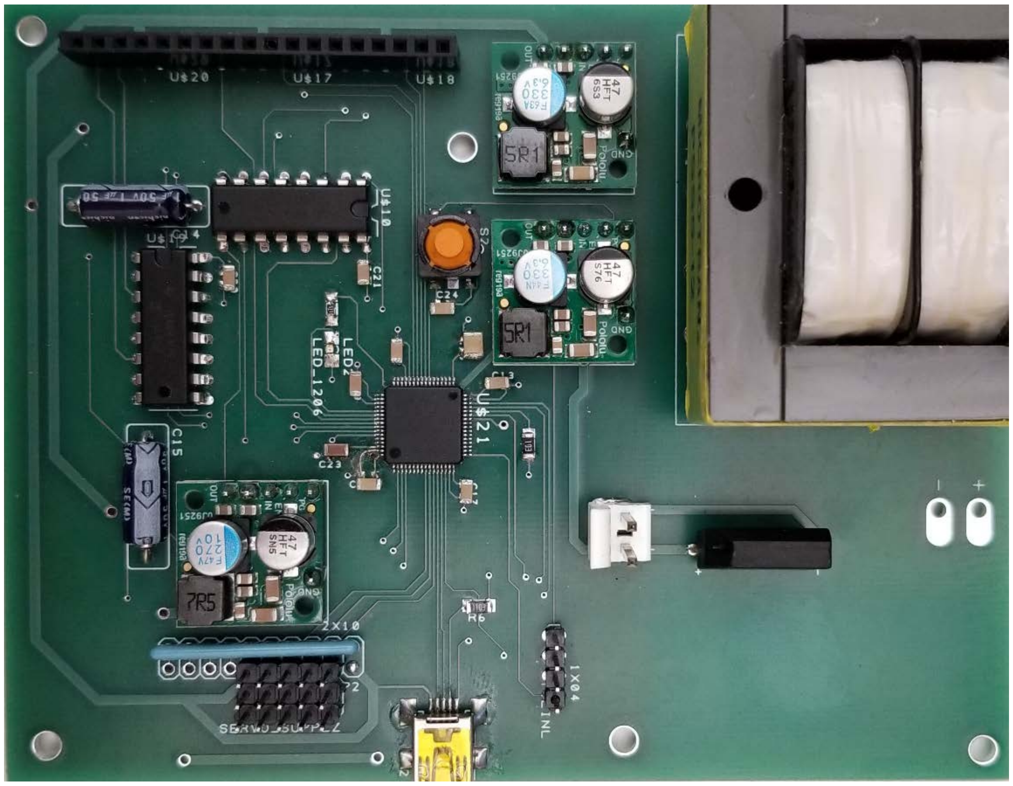


Figure 4: PCB

Mechanical Design

The arm was built using 3D printed materials, Miuzei 996R MG servo motors, Pololu 241F8X dc motors, and extruded aluminum. The 3D parts were modeled in Autodesk Inventor, then printed on a Prusa I3 printer. The models are shown in *Figure 5* and the printed pieces are shown in *Figure 6*.

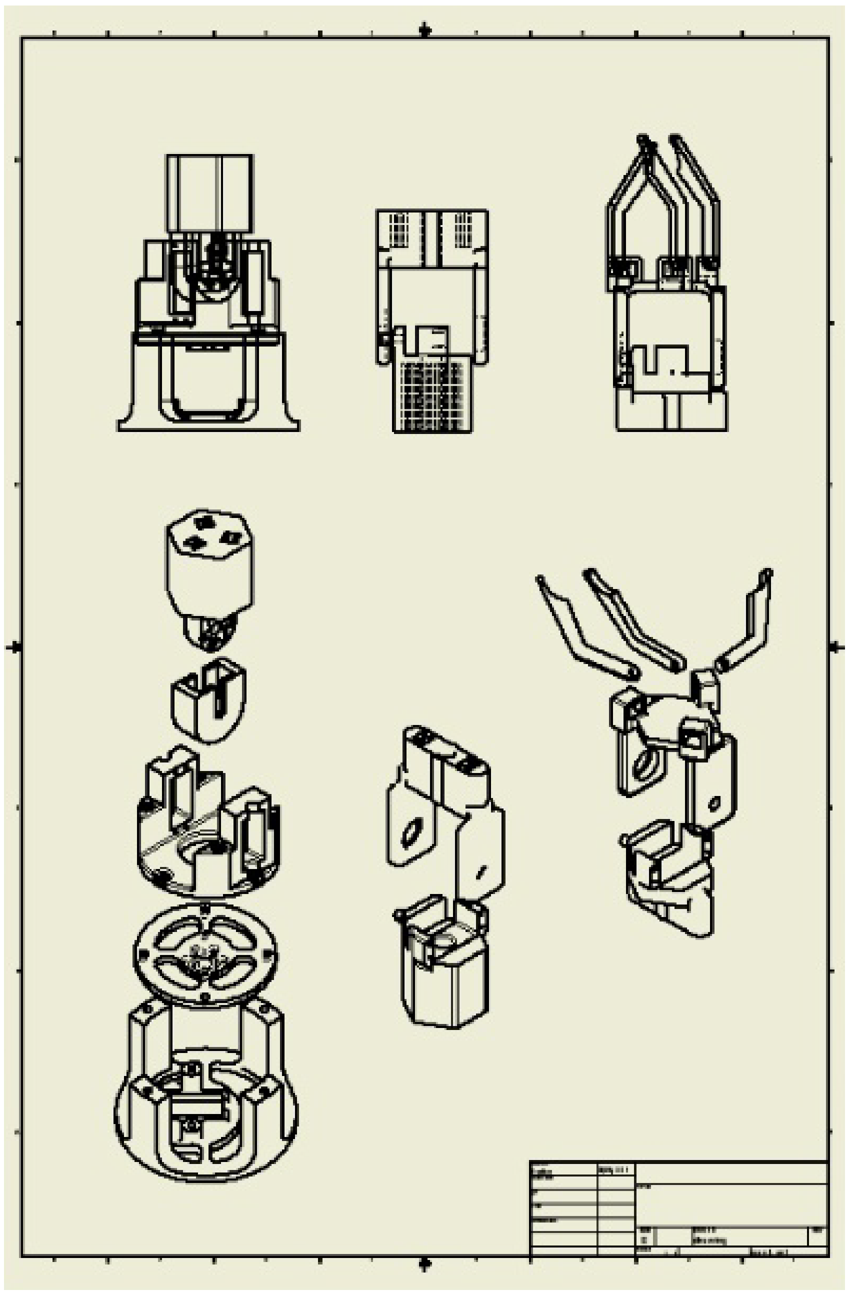


Figure 5: Arm Modeling

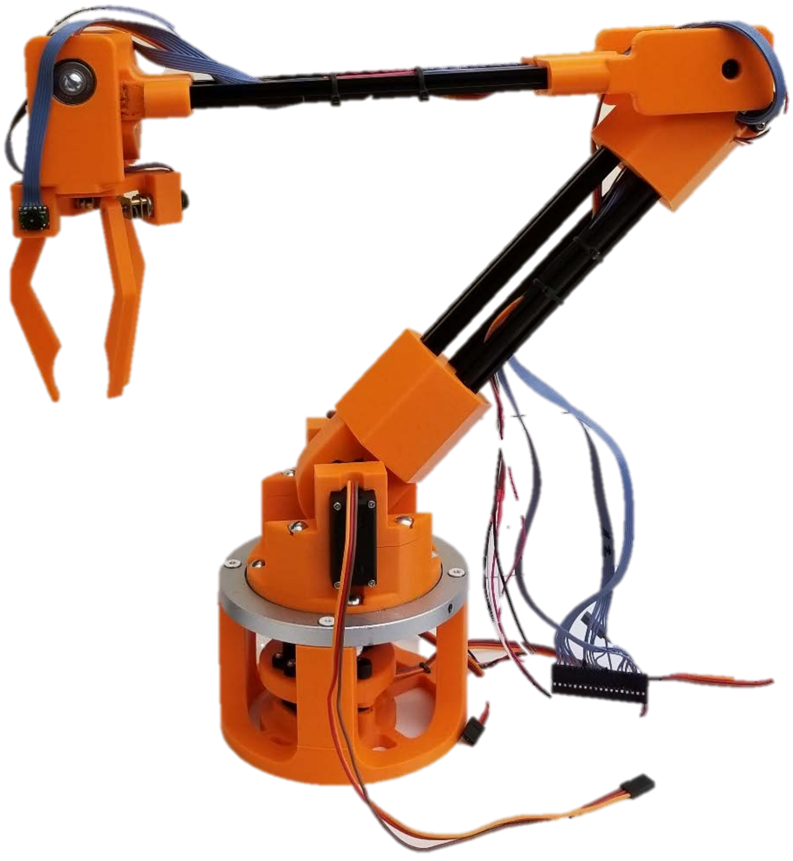


Figure 6: Robotic Arm

Software Design

To control the arm, a program was written in a Keil project in embedded C. The main source file shown in *Appendix B* runs a sequence of functions from the motor control file to move each of the motors on the arm. This file also includes the interrupt function definitions for clearing the controller’s timer interrupt flags for the pulse width modulation period cycles. The motor control source files in *Appendix A* contain all controls of the motors on the arm. The prototypes are given in the first file and the definitions in the second. The header file also includes define statements for initializing constants used for initial motor output. A *servo_t* type is defined and used for enumerating servo motors. The *MOTOR_CONTROL_INIT* function initializes all peripherals needed to output to the arm. Six external interrupt functions are defined to monitor edge detections on six I/O pins connected to the encoders on the finger motors. *Motor_Move* calculates the number of edges to occur to move a given number of degrees of rotation then turns on the motors for that number of edge counts. *Servo_MoveTo* calculates the difference in pulse width between each of the current servo states and their next states, adjusting their position incrementally at the same time for a fluid-like movement (also helps with current spikes). There are also two functions for time delaying given a number of microseconds or milliseconds using the controller's basic timers.

For visual feedback, an image recognition process was written in Octave as shown in Appendix C. The image is put through a Low Pass Filter to get edge detection. Since the resulting edges vary in intensity, a histogram is used to figure out a good point to perform a cutoff filter to show the major edges the recognition cares about.

Economic Constraints

Most of the body is made using 3D printed plastics, making it cheap and efficient in material requirements. Socially, this is an ideal product for bringing people close together through the fun of the game. Politically, people can use this as a means of compromising their opinions and beliefs when they lose a game. For health reasons, it his highly suggested any distributor of this product reinforce the importance of drinking responsibly. Manufacturability is taken care of through each component used in this product already being manufactured regularly, except for 3D printed parts which are easy to manufacture, along with software which is also easy to distribute. Sustainability is not able to be tested due to the timeframe of this project only being one year, the final product needs an extension to test durability. The final product will be safety tested. It will include a temperature sensor reading temperature in areas where heat may be an issue, such as H-bridge drivers and power supply converters. The range of motion the arm can act in will be constrained to minimalize human interaction hazards.

Results

The arm was designed, built, and programmed to grab a ball and move it around. Each of the motors is controlled via a custom PCB with an STM32F446RE microcontroller. An image of the pong layout can be given to the octave program for image recognition. The arm failed to play a game of pong nor throw a ball. Current spikes happen regularly, leaving the arm unable to move correctly. The two motors that work inverse of each other to move the base joint of the arm often come out of sink with each other, causing tension and damage to the system.

Implications for Future Work

Future modifications include changing: the dual inverse servo motors to a high-power motor with an encoder and gear box, the three finger motors to one motor and a gear system, the bottom servo motor to a stepper motor, the decision-making algorithm to a trained recurrent neural network (Wikipedia, 2019) trained via reinforcement learning (Wikipedia, 2019).

References

Semiconductor Manufacturing. (2018). *STM32L432KC*. Retrieved from ST:
https://www.st.com/en/microcontrollers/stm32l432kc.html

Wikipedia. (2019, March 28). *Recurrent Neural Network*. Retrieved from wiki:
https://en.wikipedia.org

Wikipedia. (2019, April 24). *Reinforcement Learning*. Retrieved from wiki:
https://en.wikipedia.org

Appendices

Appendix A: Arm Controls Header and Source Code Files

```c
/**
 * Author: Braden Elliott
 *
 * motor_control.h
 *        header file for the robotic arm servos
 */

#ifndef MOTOR_CONTROL_H_INCLUDED
#define MOTOR_CONTROL_H_INCLUDED

#define MIN_SERVO_PWM 1500
#define MAX_SERVO_PWM 5595
#define MOTOR_SPEED_PWM 20000
#define MOTOR_OFF_PWM 0
#define POLOLU_2385_TURN_MULTIPLIER 9.43

#define SB_StartValue 3500
#define SMJ_StartValue 3060
#define STJ_StartValue 2620

/* 50% */
#define SBJL_StartValue 3340
#define SBJR_StartValue 3140

typedef enum {
        SB,             // Servo Base
        SMJ,    // Servo Middle Joint
        STJ,    // Servo Top Joint
        SBJ             // Servo Bottom Joint
} servo_t;

typedef enum { false, true } bool_t;

/**
 *  method: MOTOR_CONTROL_INIT
 *  discription: initialize I/O, interrupts, & timers to perform motor/encoder
 */ void MOTOR_CONTROL_INIT(void);

void Motor_Clockwise(void);
void Motor_Counterclockwise(void);
void Motor_Stop(void);
void Motor_Move(int16_t);

void Servo_MoveTo(servo_t, uint16_t[4]);

void Basic_msec_Delay(uint16_t);
void Basic_usec_Delay(uint16_t);

```

```c
#endif
```

```c
/**
 * Author: Braden Elliott
 *
 * motor_control.c
 *        source file for the robotic arm motors
 */

#include "stm32f446xx.h"
#include "motor_control.h"
#include <math.h>

void MOTOR_CONTROL_INIT(void)
{
        uint32_t tmp_reg;

        /* Clock Enables */
        tmp_reg = RCC->AHB1ENR;
        tmp_reg |= RCC_AHB1ENR_GPIOAEN;
        tmp_reg |= RCC_AHB1ENR_GPIOBEN;
        tmp_reg |= RCC_AHB1ENR_GPIOCEN;
        RCC->AHB1ENR = tmp_reg;

        tmp_reg = RCC->APB1ENR;
        tmp_reg |= RCC_APB1ENR_TIM3EN;
        tmp_reg |= RCC_APB1ENR_TIM4EN;
        tmp_reg |= RCC_APB1ENR_TIM6EN;
        tmp_reg |= RCC_APB1ENR_TIM7EN;
        RCC->APB1ENR = tmp_reg;

        tmp_reg = RCC->APB2ENR;
        tmp_reg |= RCC_APB2ENR_TIM1EN;
        RCC->APB2ENR = tmp_reg;


        /* Servos: (ordered as on pcb)
         *      PA10, TIM1_3: SB   - Servo Base
         *      PA9 , TIM1_2: SMJ  - Servo Middle Joint
         *      PA8 , TIM1_1: STJ  - Servo Top Joint
         *      PB9 , TIM4_3: SBJL - Servo Bottom Joint Left
         *      PB8 , TIM4_4: SBJR - Servo Bottom Joint Right
         */
         tmp_reg = GPIOA->MODER;
         tmp_reg &= ~GPIO_MODER_MODER10;
         tmp_reg |= GPIO_MODER_MODER10_1;
         tmp_reg &= ~GPIO_MODER_MODER9;
         tmp_reg |= GPIO_MODER_MODER9_1;
         tmp_reg &= ~GPIO_MODER_MODER8;
         tmp_reg |= GPIO_MODER_MODER8_1;
         GPIOA->MODER = tmp_reg;

         tmp_reg = GPIOA->OTYPER;
         tmp_reg |= GPIO_OTYPER_OT10;
         tmp_reg |= GPIO_OTYPER_OT9;
         tmp_reg |= GPIO_OTYPER_OT8;
         GPIOA->OTYPER = tmp_reg;

         tmp_reg = GPIOA->AFR[1];
         tmp_reg &= ~GPIO_AFRH_AFSEL10;
         tmp_reg |= 1 << GPIO_AFRH_AFSEL10_Pos;
         tmp_reg &= ~GPIO_AFRH_AFSEL9;
         tmp_reg |= 1 << GPIO_AFRH_AFSEL9_Pos;
         tmp_reg &= ~GPIO_AFRH_AFSEL8;
         tmp_reg |= 1 << GPIO_AFRH_AFSEL8_Pos;
         GPIOA->AFR[1] = tmp_reg;

         tmp_reg = GPIOB->MODER;
         tmp_reg &= ~GPIO_MODER_MODER9;
         tmp_reg |= GPIO_MODER_MODER9_1;
         tmp_reg &= ~GPIO_MODER_MODER8;
         tmp_reg |= GPIO_MODER_MODER8_1;
         GPIOB->MODER = tmp_reg;

         tmp_reg = GPIOB->OTYPER;
         tmp_reg |= GPIO_OTYPER_OT9;
         tmp_reg |= GPIO_OTYPER_OT8;
         GPIOB->OTYPER = tmp_reg;

         tmp_reg = GPIOB->AFR[1];
         tmp_reg &= ~GPIO_AFRH_AFSEL9;
         tmp_reg |= 2 << GPIO_AFRH_AFSEL9_Pos;
         tmp_reg &= ~GPIO_AFRH_AFSEL8;
         tmp_reg |= 2 << GPIO_AFRH_AFSEL8_Pos;
         GPIOB->AFR[1] = tmp_reg;

         tmp_reg = TIM1->CCMR1;
         tmp_reg &= ~TIM_CCMR1_OC1M_Msk;
         tmp_reg |= 6 << TIM_CCMR1_OC1M_Pos;
         tmp_reg |= TIM_CCMR1_OC1PE;
         tmp_reg |= TIM_CCMR1_OC1FE;
         tmp_reg &= ~TIM_CCMR1_OC2M_Msk;
         tmp_reg |= 6 << TIM_CCMR1_OC2M_Pos;
         tmp_reg |= TIM_CCMR1_OC2PE;
         tmp_reg |= TIM_CCMR1_OC2FE;
         TIM1->CCMR1 = tmp_reg;

         tmp_reg = TIM1->CCMR2;
         tmp_reg &= ~TIM_CCMR2_OC3M_Msk;
         tmp_reg |= 6 << TIM_CCMR2_OC3M_Pos;
         tmp_reg |= TIM_CCMR2_OC3PE;
         tmp_reg |= TIM_CCMR2_OC3FE;
         TIM1->CCMR2 = tmp_reg;

         tmp_reg = TIM1->CCER;
         tmp_reg |= TIM_CCER_CC1E;
         tmp_reg |= TIM_CCER_CC2E;
         tmp_reg |= TIM_CCER_CC3E;
         TIM1->CCER = tmp_reg;

         TIM1->PSC = 7;
         TIM1->ARR = 40000;
         TIM1->CCR1 = STJ_StartValue;
         TIM1->CCR2 = SMJ_StartValue;
         TIM1->CCR3 = SB_StartValue;
         TIM1->BDTR |= TIM_BDTR_MOE;

         tmp_reg = TIM1->CR1;
         tmp_reg |= TIM_CR1_ARPE;
         tmp_reg |= TIM_CR1_CEN;
         TIM1->CR1 = tmp_reg;

         TIM1->EGR |= TIM_EGR_UG;

         tmp_reg = TIM4->CCMR2;
         tmp_reg &= ~TIM_CCMR2_OC3M_Msk;
         tmp_reg |= 6 << TIM_CCMR2_OC3M_Pos;
         tmp_reg |= TIM_CCMR2_OC3PE;
         tmp_reg |= TIM_CCMR2_OC3FE;
         tmp_reg &= ~TIM_CCMR2_OC4M_Msk;
         tmp_reg |= 6 << TIM_CCMR2_OC4M_Pos;
         tmp_reg |= TIM_CCMR2_OC4PE;
         tmp_reg |= TIM_CCMR2_OC4FE;
         TIM4->CCMR2 = tmp_reg;

         tmp_reg = TIM4->CCER;
         tmp_reg |= TIM_CCER_CC3E;
         tmp_reg |= TIM_CCER_CC4E;
         TIM4->CCER = tmp_reg;

         TIM4->PSC = 7;
         TIM4->ARR = 40000;
         TIM4->CCR3 = SBJL_StartValue;
         TIM4->CCR4 = SBJR_StartValue;

         tmp_reg = TIM4->CR1;
         tmp_reg |= TIM_CR1_ARPE;
         tmp_reg |= TIM_CR1_CEN;
         TIM4->CR1 = tmp_reg;


        /* Pololu Motor: MC
         *      PB6 , TIM4_1: MC_M2 - Motor C M2 on hand
         *      PB7 , TIM4_2: MC_M1 - Motor C M1 on hand
         *      PB15, GP_Out: MC_En - Motor C Enable
         *      PC4 , EXTI     : MC_OB - Motor C OutB
         *      PC5 , EXTI     : MC_OA - Motor C OutA
         */

         tmp_reg = TIM4->CCMR1;
         tmp_reg &= ~TIM_CCMR1_OC1M_Msk;
         tmp_reg |= 6 << TIM_CCMR1_OC1M_Pos;
         tmp_reg |= TIM_CCMR1_OC1PE;
         tmp_reg |= TIM_CCMR1_OC1FE;
         tmp_reg &= ~TIM_CCMR1_OC2M_Msk;
         tmp_reg |= 6 << TIM_CCMR1_OC2M_Pos;
         tmp_reg |= TIM_CCMR1_OC2PE;
         tmp_reg |= TIM_CCMR1_OC2FE;
         TIM4->CCMR1 = tmp_reg;

         tmp_reg = TIM4->CCER;
         tmp_reg |= TIM_CCER_CC1E;
         tmp_reg |= TIM_CCER_CC2E;
         TIM4->CCER = tmp_reg;

         TIM4->CCR1 = MOTOR_OFF_PWM;
         TIM4->CCR2 = MOTOR_OFF_PWM;

         NVIC_EnableIRQ(TIM4_IRQn);
         TIM4->DIER |= TIM_DIER_CC1IE;
         TIM4->DIER |= TIM_DIER_CC2IE;

         tmp_reg = GPIOB->MODER;
         tmp_reg &= ~GPIO_MODER_MODER15;
         tmp_reg |= GPIO_MODER_MODER15_0;
         tmp_reg &= ~GPIO_MODER_MODER6;
         tmp_reg |= GPIO_MODER_MODER6_1;
         tmp_reg &= ~GPIO_MODER_MODER7;
         tmp_reg |= GPIO_MODER_MODER7_1;
         GPIOB->MODER = tmp_reg;

         tmp_reg = GPIOB->OTYPER;
         tmp_reg |= GPIO_OTYPER_OT15;
         GPIOB->OTYPER = tmp_reg;

         tmp_reg = GPIOB->AFR[0];
         tmp_reg &= ~GPIO_AFRL_AFSEL6;
         tmp_reg |= 2 << GPIO_AFRL_AFSEL6_Pos;
         tmp_reg &= ~GPIO_AFRL_AFSEL7;
         tmp_reg |= 2 << GPIO_AFRL_AFSEL7_Pos;
         GPIOB->AFR[0] = tmp_reg;

         tmp_reg = GPIOC->MODER;
         tmp_reg &= ~GPIO_MODER_MODER4;
         tmp_reg &= ~GPIO_MODER_MODER5;
         GPIOC->MODER = tmp_reg;

         NVIC_EnableIRQ(EXTI4_IRQn);
         NVIC_EnableIRQ(EXTI9_5_IRQn);

         tmp_reg = EXTI->IMR;
         tmp_reg |= EXTI_EMR_EM4;
         tmp_reg |= EXTI_EMR_EM5;
         EXTI->IMR = tmp_reg;

         tmp_reg = EXTI->RTSR;
         tmp_reg |= EXTI_RTSR_TR4;
         tmp_reg |= EXTI_RTSR_TR5;
         EXTI->RTSR = tmp_reg;

         tmp_reg = EXTI->FTSR;
```

```c
        tmp_reg |= EXTI_FTSR_TR4;
        tmp_reg |= EXTI_FTSR_TR5;
        EXTI->FTSR = tmp_reg;

        RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
        SYSCFG->EXTICR[1] |= SYSCFG_EXTICR2_EXTI4_PC;
        SYSCFG->EXTICR[1] |= SYSCFG_EXTICR2_EXTI5_PC;


       /* Pololu Motor: MB
        *      PB0 , TIM3_3: MB_M2 - Motor B M2 on hand
        *      PB1 , TIM3_4: MB_M1 - Motor B M1 on hand
        *      PB14, GP_Out: MB_En - Motor B Enable
        *      PC3 , EXTI      : MB_OB - Motor B OutB
        *      PC2 , EXTI      : MB_OA - Motor B OutA
        */

        tmp_reg = GPIOB->MODER;
        tmp_reg &= ~GPIO_MODER_MODER0;
        tmp_reg |= GPIO_MODER_MODER0_1;
        tmp_reg &= ~GPIO_MODER_MODER1;
        tmp_reg |= GPIO_MODER_MODER1_1;
        GPIOB->MODER = tmp_reg;

        tmp_reg = GPIOB->OTYPER;
        tmp_reg |= GPIO_OTYPER_OT14;
        GPIOB->OTYPER = tmp_reg;

        tmp_reg = GPIOB->AFR[0];
        tmp_reg &= ~GPIO_AFRL_AFSEL0;
        tmp_reg |= 2 << GPIO_AFRL_AFSEL0_Pos;
        tmp_reg &= ~GPIO_AFRL_AFSEL1;
        tmp_reg |= 2 << GPIO_AFRL_AFSEL1_Pos;
        GPIOB->AFR[0] = tmp_reg;

        tmp_reg = TIM3->CCMR2;
        tmp_reg &= ~TIM_CCMR2_OC3M_Msk;
        tmp_reg |= 6 << TIM_CCMR2_OC3M_Pos;
        tmp_reg |= TIM_CCMR2_OC3PE;
        tmp_reg |= TIM_CCMR2_OC3FE;
        tmp_reg &= ~TIM_CCMR2_OC4M_Msk;
        tmp_reg |= 6 << TIM_CCMR2_OC4M_Pos;
        tmp_reg |= TIM_CCMR2_OC4PE;
        tmp_reg |= TIM_CCMR2_OC4FE;
        TIM3->CCMR2 = tmp_reg;

        tmp_reg = TIM3->CCER;
        tmp_reg |= TIM_CCER_CC3E;
        tmp_reg |= TIM_CCER_CC4E;
        TIM3->CCER = tmp_reg;

        TIM3->PSC = 7;
        TIM3->ARR = 40000;
        TIM3->CCR3 = MOTOR_OFF_PWM;
        TIM3->CCR4 = MOTOR_OFF_PWM;

        tmp_reg = TIM3->CR1;
        tmp_reg |= TIM_CR1_ARPE;
        tmp_reg |= TIM_CR1_CEN;
        TIM3->CR1 = tmp_reg;

        NVIC_EnableIRQ(TIM3_IRQn);
        TIM3->DIER |= TIM_DIER_CC3IE;
        TIM3->DIER |= TIM_DIER_CC4IE;

        tmp_reg = GPIOC->MODER;
        tmp_reg &= ~GPIO_MODER_MODER2;
        tmp_reg &= ~GPIO_MODER_MODER3;
        GPIOC->MODER = tmp_reg;

        NVIC_EnableIRQ(EXTI2_IRQn);
        NVIC_EnableIRQ(EXTI3_IRQn);

        tmp_reg = EXTI->IMR;
        tmp_reg |= EXTI_EMR_EM2;
        tmp_reg |= EXTI_EMR_EM3;
        EXTI->IMR = tmp_reg;

        tmp_reg = EXTI->RTSR;
        tmp_reg |= EXTI_RTSR_TR2;
        tmp_reg |= EXTI_RTSR_TR3;
        EXTI->RTSR = tmp_reg;

        tmp_reg = EXTI->FTSR;
        tmp_reg |= EXTI_FTSR_TR2;
        tmp_reg |= EXTI_FTSR_TR3;
        EXTI->FTSR = tmp_reg;

        RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
        SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI2_PC;
        SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI3_PC;


       /* Pololu Motor: MA
        *      PA7 , TIM3_1: MA_M2 - Motor A M2 on hand
        *      PA6 , TIM3_2: MA_M1 - Motor A M1 on hand
        *      PB13, GP_Out: MA_En - Motor A Enable
        *      PC0 , EXTI      : MA_OB - Motor A OutB
        *      PC1 , EXTI      : MA_OA - Motor A OutA
        */

        tmp_reg = GPIOA->MODER;
        tmp_reg &= ~GPIO_MODER_MODER6;
        tmp_reg |= GPIO_MODER_MODER6_1;
        tmp_reg &= ~GPIO_MODER_MODER7;
        tmp_reg |= GPIO_MODER_MODER7_1;
        GPIOA->MODER = tmp_reg;

        tmp_reg = GPIOB->OTYPER;
        tmp_reg |= GPIO_OTYPER_OT13;
        GPIOB->OTYPER = tmp_reg;

```

```c
        tmp_reg = GPIOA->AFR[0];
        tmp_reg &= ~GPIO_AFRL_AFSEL6;
        tmp_reg |= 2 << GPIO_AFRL_AFSEL6_Pos;
        tmp_reg &= ~GPIO_AFRL_AFSEL7;
        tmp_reg |= 2 << GPIO_AFRL_AFSEL7_Pos;
        GPIOA->AFR[0] = tmp_reg;

        tmp_reg = TIM3->CCMR1;
        tmp_reg &= ~TIM_CCMR1_OC1M_Msk;
        tmp_reg |= 6 << TIM_CCMR1_OC1M_Pos;
        tmp_reg |= TIM_CCMR1_OC1PE;
        tmp_reg |= TIM_CCMR1_OC1FE;
        tmp_reg &= ~TIM_CCMR1_OC2M_Msk;
        tmp_reg |= 6 << TIM_CCMR1_OC2M_Pos;
        tmp_reg |= TIM_CCMR1_OC2PE;
        tmp_reg |= TIM_CCMR1_OC2FE;
        TIM3->CCMR1 = tmp_reg;

        tmp_reg = TIM3->CCER;
        tmp_reg |= TIM_CCER_CC1E;
        tmp_reg |= TIM_CCER_CC2E;
        TIM3->CCER = tmp_reg;

        TIM3->CCR1 = MOTOR_OFF_PWM;
        TIM3->CCR2 = MOTOR_OFF_PWM;

        NVIC_EnableIRQ(TIM3_IRQn);
        TIM3->DIER |= TIM_DIER_CC1IE;
        TIM3->DIER |= TIM_DIER_CC2IE;

        tmp_reg = GPIOC->MODER;
        tmp_reg &= ~GPIO_MODER_MODER0;
        tmp_reg &= ~GPIO_MODER_MODER1;
        GPIOC->MODER = tmp_reg;

        NVIC_EnableIRQ(EXTI0_IRQn);
        NVIC_EnableIRQ(EXTI1_IRQn);

        tmp_reg = EXTI->IMR;
        tmp_reg |= EXTI_EMR_EM0;
        tmp_reg |= EXTI_EMR_EM1;
        EXTI->IMR = tmp_reg;

        tmp_reg = EXTI->RTSR;
        tmp_reg |= EXTI_RTSR_TR0;
        tmp_reg |= EXTI_RTSR_TR1;
        EXTI->RTSR = tmp_reg;

        tmp_reg = EXTI->FTSR;
        tmp_reg |= EXTI_FTSR_TR0;
        tmp_reg |= EXTI_FTSR_TR1;
        EXTI->FTSR = tmp_reg;

        RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;
        SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI0_PC;
        SYSCFG->EXTICR[0] |= SYSCFG_EXTICR1_EXTI1_PC;
```

```c
}


void Motor_Stop(void)
{
        GPIOB->ODR &= ~(1<<15);

        TIM4->CR1  &= ~TIM_CR1_CEN;
        TIM4->EGR       |= TIM_EGR_UG;

        GPIOB->ODR &= ~(1<<14);
        TIM3->CR1  &= ~TIM_CR1_CEN;
        TIM3->EGR       |= TIM_EGR_UG;

        GPIOB->ODR &= ~(1<<13);

        TIM3->CR1  &= ~TIM_CR1_CEN;
        TIM3->EGR       |= TIM_EGR_UG;
}

long edge_counts_A, edge_counts_B, edge_counts_C;
void Motor_Move(int16_t deg)
{
        edge_counts_A = (POLOLU_2385_TURN_MULTIPLIER*(long)fabs((float)deg));
        edge_counts_B = (POLOLU_2385_TURN_MULTIPLIER*(long)fabs((float)deg));
        edge_counts_C = (POLOLU_2385_TURN_MULTIPLIER*(long)fabs((float)deg));

        GPIOB->ODR |= 1<<15;
        TIM4->CR1  |= TIM_CR1_CEN;

        GPIOB->ODR |= 1<<14;
        TIM3->CR1  |= TIM_CR1_CEN;

        GPIOB->ODR |= 1<<13;
        TIM3->CR1  |= TIM_CR1_CEN;

        if(deg > 0)
                Motor_Clockwise();
        else if(deg == 0)
                return;
        else
                Motor_Counterclockwise();

        while(edge_counts_A > 0 && edge_counts_B > 0 && edge_counts_C > 0);

        TIM3->CCR2 = MOTOR_OFF_PWM;
        TIM3->CCR1 = MOTOR_OFF_PWM;

        TIM3->CCR4 = MOTOR_OFF_PWM;
        TIM3->CCR3 = MOTOR_OFF_PWM;

        TIM4->CCR2 = MOTOR_OFF_PWM;
        TIM4->CCR1 = MOTOR_OFF_PWM;
}

void Motor_Counterclockwise(void)
{
        TIM4->CCR2 = MOTOR_SPEED_PWM;
        TIM4->CCR1 = MOTOR_OFF_PWM;

        TIM3->CCR2 = MOTOR_SPEED_PWM;
        TIM3->CCR1 = MOTOR_OFF_PWM;

        TIM3->CCR3 = MOTOR_SPEED_PWM;
        TIM3->CCR4 = MOTOR_OFF_PWM;
}

void Motor_Clockwise(void)
{
        TIM4->CCR1 = MOTOR_SPEED_PWM;
        TIM4->CCR2 = MOTOR_OFF_PWM;

        TIM3->CCR1 = MOTOR_SPEED_PWM;
        TIM3->CCR2 = MOTOR_OFF_PWM;

        TIM3->CCR4 = MOTOR_SPEED_PWM;
        TIM3->CCR3 = MOTOR_OFF_PWM;
}

void EXTI0_IRQHandler(void)
{
        if(EXTI->PR & EXTI_PR_PR0)
        {
                edge_counts_A--;
                EXTI->PR |= EXTI_PR_PR0;
        }
}

void EXTI1_IRQHandler(void)
{
        if(EXTI->PR & EXTI_PR_PR1)
        {
                edge_counts_A--;
                EXTI->PR |= EXTI_PR_PR1;
        }
}

void EXTI2_IRQHandler(void)
{
        if(EXTI->PR & EXTI_PR_PR2)
        {
                edge_counts_B--;
                EXTI->PR |= EXTI_PR_PR2;
        }
}

void EXTI3_IRQHandler(void)
{
```

```c
        if(EXTI->PR & EXTI_PR_PR3)
        {
                edge_counts_B--;
                EXTI->PR |= EXTI_PR_PR3;
        }
}

void EXTI4_IRQHandler(void)
{
        if(EXTI->PR & EXTI_PR_PR4)
        {
                edge_counts_C--;
                EXTI->PR |= EXTI_PR_PR4;
        }
}

void EXTI9_5_IRQHandler(void)
{
        if(EXTI->PR & EXTI_PR_PR5)
        {
                edge_counts_C--;
                EXTI->PR |= EXTI_PR_PR5;
        }
}

void Servo_MoveTo(servo_t s, uint16_t pos[4])
{
        uint16_t new_pos[4];
        uint16_t tmp[4];
        bool_t dir[4];
        bool_t flag[] = {true, true, true, true};


        new_pos[SB]     = (40*pos[SB])  + MIN_SERVO_PWM;
        new_pos[SMJ]    = (40*pos[SMJ]) + MIN_SERVO_PWM;
        new_pos[STJ]    = (40*pos[STJ]) + MIN_SERVO_PWM;
        new_pos[SBJ]    = (40*pos[SBJ]) + MIN_SERVO_PWM;

        tmp[SB]         = TIM1->CCR3 & 0xFFFF;
        tmp[SMJ]        = TIM1->CCR2 & 0xFFFF;
        tmp[STJ]        = TIM1->CCR1 & 0xFFFF;

        dir[SB]         = (tmp[SB] > new_pos[SB])      ? true: false;
        dir[SMJ]        = (tmp[SMJ] > new_pos[SMJ]) ? true: false;
        dir[STJ]        = (tmp[STJ] > new_pos[STJ]) ? true: false;

        while(flag[SB] && flag[SMJ] && flag[STJ])
        {
                /* increment & decrement pulse widths */
                if(flag[SB])
                {
                        TIM1->CCR3 = tmp[SB]    | 0xFFFF0000;
                        tmp[SB]         += (dir[SB])    ? -50: 50;
                }
                if(flag[SMJ])
                {
                        TIM1->CCR2 = tmp[SMJ] | 0xFFFF0000;
                        tmp[SMJ]       += (dir[SMJ])  ? -50: 50;
                }
                if(flag[STJ])
                {
                        TIM1->CCR1 = tmp[STJ] | 0xFFFF0000;
                        tmp[STJ]       += (dir[STJ]) ? -50: 50;
                }

                /* check for pulse width within final range & update flags */
                if(tmp[SB] < new_pos[SB]+50 && tmp[SB] > new_pos[SB]-50)
                {
                        TIM1->CCR3 = new_pos[SB] | 0xFFFF0000;
                        flag[SB] = false;
                }
                if(tmp[SMJ] < new_pos[SMJ]+50 && tmp[SMJ] > new_pos[SMJ]-50)
                {
                        TIM1->CCR2 = new_pos[SMJ] | 0xFFFF0000;
                        flag[SMJ] = false;
                }
                if(tmp[STJ] < new_pos[STJ]+50 && tmp[STJ] > new_pos[STJ]-50)
                {
                        TIM1->CCR1 = new_pos[STJ] | 0xFFFF0000;
                        flag[STJ] = false;
                }

                /* Delay for slower movements & to avoid current spikes */
                Basic_msec_Delay(40);
        }

}

void Basic_msec_Delay(uint16_t ms)
{
        TIM6->CR1 &= ~1;        // counter disable
        TIM6->PSC = 15999;      // 16MHz/(15999+1) = 1ms
        TIM6->ARR = ms;         // counter counts 'ms' times
        TIM6->CR1 |= 1;         // counter enable
        TIM6->EGR |= 1;         // update generation

        TIM6->SR &= ~1;         // clear update interrupt flag
        while(!(TIM6->SR & 1)); // wait for update interrupt flag
        TIM6->CR1 &= ~1;        // counter disable
}

void Basic_usec_Delay(uint16_t us)
{
        TIM7->CR1 &= ~1;        // counter disable
        TIM7->PSC = 15;         // 16MHz/(15+1) = 1us
        TIM7->ARR = us;         // counter counts 'us' times
        TIM7->CR1 |= 1;         // counter enable
```

```c
        TIM7->EGR |= 1;         // update generation

        TIM7->SR &= ~1;         // clear update interrupt flag
        while(!(TIM7->SR & 1)); // wait for update interrupt flag
        TIM7->CR1 &= ~1;        // counter disable
}
```

Appendix B: Main Source Code File for Arm Movement Demo

```c
/**
 * Author: Braden Elliott
 *
 * main.c
 *              main source file for the robotic arm control program
 *              this program performs a demo of the arm using the arm source control files
 */

#include "stm32f446xx.h"
#include "motor_control.h"

/**
 *  method: Init
 *  discription: initialize I/O, timers, interrupts, etc.
 */
void Init()
{

}

int main()
{
        Init();
        MOTOR_CONTROL_INIT();
        Basic_msec_Delay(1000);

        uint16_t moves[] = {30, 52, 50, 50};
        Servo_MoveTo(SB, moves);
        moves[SB] = 50;
        Servo_MoveTo(SB, moves);
        Servo_MoveTo(STJ, moves);
        moves[STJ] = 28;
        Servo_MoveTo(STJ, moves);
        Servo_MoveTo(SBJ, moves);
        Servo_MoveTo(SMJ, moves);
        Basic_msec_Delay(500);
        Motor_Move(10);
        Basic_msec_Delay(250);
        Motor_Move(-10);

        while(1);
}

void TIM3_IRQHandler(void)
{
        uint8_t status = TIM3->SR & 0xFF;
        if(status & 0x02)
                TIM3->SR &= 0xE1A0;
}

void TIM4_IRQHandler(void)
{
        uint8_t status = TIM4->SR & 0xFF;
        if(status & 0x02)
                TIM4->SR &= 0xE1A0;
}

void TIM9_5_IRQHandler(void)
{
        uint8_t status = TIM5->SR & 0xFF;
        if(status & 0x02)
                TIM5->SR &= 0xE1A0;
}
```

Appendix C: Image Recognition Code File

```
% Braden Elliott
% pong cup & ball recognition

clear all

function z = ue_sobel(f, T)
  hh = [-1 0 1; -2 0 2; -1 0 1];
  hv = hh’;

   dh = conv2(f, hh, 'same');
   dv = conv2(f, hv, 'same');

   g = sqrt(dh.^2 + dv.^2);

   z = g>T;
endfunction

x = imrotate(rgb2gray(imread('pongtest04.jpg')), 270);
[M N] = size(x);
a = x((M/3):(M-(M/6)), 1:N);

% Butterworth LPF

[M N] = size(a);
P = 2*M;
Q = 2*N;

X = fftshift(fft2(a, P, Q));
D0 = 100;  % lower D0 to see more blurry, higher for less blurry
D0SQ = D0*D0;
n = 4;
H = 1 - (1)./(1 + (d_squared_ctr(P, Q)/D0SQ).^n);  % edges
Y = X .* H;
y_pad = uint8(abs(real(ifft2(ifftshift(Y)))));
y = y_pad(1:M,1:N);
figure
imshow(y)

w = y;
w(w<50) = 255;
w(y>50) = 0;
figure
```

```
imshow(w)

figure
hist(w(:), 50)
```