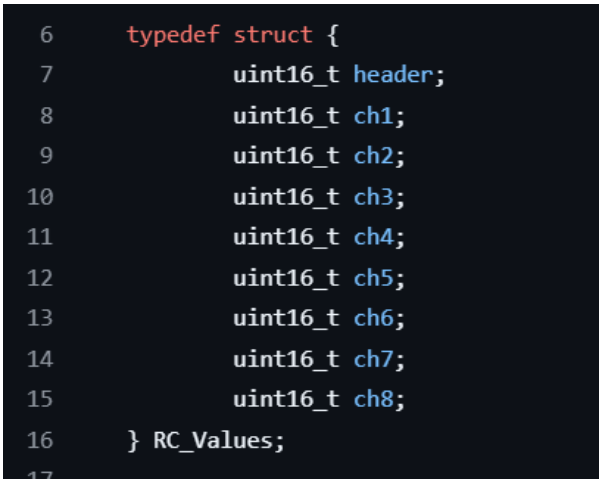
4DS4 Project 1 Report:
Towards a More Realistic System

Johnnathan Gershkovich - 400408809
Joshua Umansky - 400234265
Shanzeb Immad - 400382928

Name	Contribution
Johnnathan Gershkovich	Aided in the implementations for Led Component, Motor Component and RX Component, Wrote report
Joshua Umansky	Aided in the implementations for Led Component, Motor Component and RX Component, Wrote report
Shanzeb Immad	Aided in the implementations for Led Component, Motor Component and RX Component, Wrote report

Rx Component

The RC_Receiver_Component.c file handles most of the code's functionality by reading the channels and enqueuing to the queues used by the other files. Setup of this file included adding struct to read the channels received from the controllers. A RC task was also made within the setup that handles all the interactions of this component with the other components.



Rc_Values structure

The code for the rc pin setups and UART setup was made by referring to previous labs that used UART and RC. The RC task begins by initializing needed variables such as the mode, speed, angle and direction setting them to 0. Once the while loop begins and the channels are being observed, the value of channel 5 is first observed which correlates to the switch used to control direction. Forward being 1 while reverse is -1. Next the switch that controls the speed mode is observed as channel 6. The three modes control the max speed being 50%, 75% and 100% of the normal speed. The if statements looking at channel 6 sets the mode variable to 1,2 or 3 and normalizes the speed equations depending on the direction and mode.

```
if(rc_values.ch5 == 1000)
        {direction = 1;}
if(rc_values.ch5 == 2000)
        {direction = -1;}

if(rc_values.ch6 == 1000){ //max 100% speed 0-100
        mode = 2;
        speed = direction * (rc_values.ch3 - 1000) / 10;
} else if(rc_values.ch6 == 1500){ //max 75% speed 0-75
        mode = 1;
        speed = direction * (rc_values.ch3 - 1000) * 75 / 1000;
} else if(rc_values.ch6 == 2000){ //max 50% speed 0-50
        mode = 0;
        speed = direction * (rc_values.ch3 - 1000) * 50 / 1000;
}
```

Once the mode and speed are determined, channel 1 is observed as it controls the servo motor. Since all the channels only send values between 1000 and 2000, the following equation is used to normalize it to be between [-180 and 180].

$$angle = (-180 + 0.36(channel\,1 - 1000))/2$$

The final aspect of this task is utilizing the queues so that the other components can utilize these values. All the queues in our group's code are globally initialized and set to size 1 with the size of int. The reason we set the to the size of 1 is that if a value is enqueued and another value wants to enter the queue, the xQueueReceive is called right away so there is no buffering.

```
printf("Speed: %d\n", speed);
printf("Mode: %d\n", mode);
printf("Angle: %d\n", angle);
//printf("Before first queue");
status = xQueueSendToBack(motor_queue, (void*) &speed, portMAX_DELAY);
//printf("Before second queue");
status = xQueueSendToBack(angle_queue, (void*) &angle, portMAX_DELAY);
printf("Before third queue\n");
status = xQueueSendToBack(led_queue, (void*) &mode, portMAX_DELAY);
```

Motor Component

The motor component was implemented using the building blocks learned in lab 1 in regards to operating the drive motors, and the servo motor to control the turning angle. In both situations, we take input from the respective channels from the controller (Channel 1 Servo motor, Channel 3 Drive motor).

The duty cycle motor formula is defined as the following;

$$Duty\,Cycle = (speed * \frac{0.025}{100}) + 0.0615$$

Where the value of speed is a value between -100 and 100, depending on the mode currently being operated in, and passed as a parameter from the task creation, which is set

ultimately by the receiver, which is passed into the motor queue. The significance of the 0.0615 is used in order to account for a minute value of drift in the controller.

The duty cycle servo formula is defined as the following;

$$Duty\ Cycle\ = \frac{(angle + 90)*0.05}{180}\ +\ 0.05$$

With angle being any input angle from -90 to 90 degrees. -90 degrees represents turning full right, 0 being straight and 90 being full right. This value of angle is received as a parameter similar to speed in the motor control, which is a result of the angle_queue.

```
void motorTask(void* pvParameters)
{
	int dcinput = 0;
	int speed = 0;
	BaseType_t status;
	float dutyCycle_DC;
	//Motor task implementation
	while(1){
		status = xQueueReceive(motor_queue, (void *)&speed, portMAX_DELAY);
		if(status != pdPASS){
			PRINTF("QUEUE receive failed!\r\n");
			while(1);
		}
		dutyCycle_DC = speed * 0.025f/100.0f + 0.0615;
		updatePWM_dutyCycle(FTM_CHANNEL_DC_MOTOR, dutyCycle_DC);
		FTM_SetSoftwareTrigger(FTM_MOTORS, true);
	}

}
```

motorTask implementation, positionTask similarly defined

LED Component

The LED component is fairly simple as all the pin setups are used from lab 0 and 1. This file globally creates the LED queue and a LED task. The task dequeues the led queue which holds the mode value. Depending on the mode, the RGB for the led is changed, a green light is displayed if 50% speed mode is chosen, a yellow light is made for the 75% speed mode and lastly a red light at 100% speed mode. Once these values are set the FTM PWM update lines are called for each RGB channel updating it to the mode's colour values.

```
while(1){

        status = xQueueReceive(led_queue, (void *)&mode, portMAX_DELAY);
        //printf("dequeued!\n");
        int red = 0;
        int blue = 0;
        int green = 0;


        if(mode == 0){ //Green
                green = 100;
                red = 0;
                blue = 0;
        } else if(mode == 1){ //Yellow
                green = 100;
                red = 100;
                blue = 0;
        } else if(mode == 2){ //Red
                red = 100;
                green = 0;
                blue = 0;
        }

        FTM_UpdatePwmDutycycle(FTM3, FTM_RED_CHANNEL, kFTM_EdgeAlignedPwm, red);
        FTM_UpdatePwmDutycycle(FTM3, FTM_BLUE_CHANNEL, kFTM_EdgeAlignedPwm, blue);
        FTM_UpdatePwmDutycycle(FTM3, FTM_GREEN_CHANNEL, kFTM_EdgeAlignedPwm, green);
        FTM_SetSoftwareTrigger(FTM3, true);
}
```

LedTask implementation