Final Project Report: Story of Jankbot

Chris Lutze

Jennifer Hawthorne

Tristan Thompson

ME 405-03

3/16/15

Table of Contents

1.0 Introduction

The goal of this project was to create an autonomous robot to shoot a Nerf gun or ping pong ball at an opposing duelist located approximately 16 feet away from the user. The motivation that drove the final design for this robot was to acquire and engage the target as quickly and accurately as possible. This was the foundational philosophy while designing our team's robot, lovingly referred to as the Jankbot NERF gun platform. The Jankbot system utilized a custom optic that focused infrared light on an array of phototransistors to locate targets. Once a target was located, the Jankbot system would ideally aim at the opposing duelist and fire a single shot.

In a real dueling scenario, this system would allow the user to brandish extra weapons with his or her free hands, while the Jankbot platform suppressed the target autonomously. The ideal customer for this project would be someone who loves to engage in duels involving NERF guns, swords, and other products. This demographic may include children as well as adults, so safety was a paramount consideration during the design process. Owners of the Jankbot system would have a significant advantage over their opponents in a duel, thus this product would be quite appealing to them. At the time of the Jankbot system's creation, our group had learned to write code for an analog to digital (A/D) convertor, motor controller, encoder reader, and position controller. All of the above skills were implemented during the design of the Jankbot system.

Unfortunately, our group was not able to achieve the ideal results detailed above. Our group was able to successfully create a working infrared sensor array and created software that successfully controlled DC motors using input from optical encoders. Before the deadline, our team began to implement the control loop that would spin the gun 180 degrees, acquire, and engage the target, but was unable to finish the code by the deadline due to hardware and software errors. The errors encountered are detailed in the following sections of this report.

2.0 Specifications

The Jankbot system could be started by the user, but was to be completely autonomous following this input. The system had to start facing 180 degrees away from its target and then rotate to face the target. The target was an infrared light source on a 12 inch by 12 inch wooden shield. Once it turned to face the target, Jankbot could scan to find IR light, lock onto it, and fire. This was achieved through use of a 2-axis motor control system using infrared and encoder sensor feedback.

Our group decided to make Jankbot to weigh less than 20 pounds so that it could be picked up and moved easily when needed. It also needed to rotate a full 360 degrees in order to meet the design requirements. This was due to the fact that the target may be at a location slightly greater than 180 degrees from Jankbot. Jankbot also needed to aim approximately three feet vertically from the table surface, so as to hit a wider range of targets. The system required accuracy of six inches in any direction away from the infrared light source at a position 16 feet away from the target. This would allow Jankbot to hit the 12 inch x 12 inch board on which the infrared light source was centered. Lastly, the optical sensor array would need to be able to focus and detect light from a bulb located 16 feet downrange of the array. These specifications and requirements are summarized in Table 1 below:

Table 1: Design Specifications and Requirements Summary- This table summarizes the basic specifications and requirements used to design Jankbot

Requirements	Specification
Weight	< 20 lb
Rotation	180 ± 5 degrees
Tilt	Able to hit targets 0 to 3 ft above Jankbot
Accuracy	Hit the within 6 inches of the infrared light at a distance of 16 feet
Sensor Fidelity	Able to focus infrared light from a source 16 feet away

3.0 Design Development: Hardware Development

The original concept for Jankbot was created during the early phases of the project during class. Our team generated sketches depicting two different visions for the final mechanical design (see Appendix A); however, the final design differed significantly from both concepts.

3.1 Pan Axis Design

Our team's initial plan was to use Lego® Technic gears to create a large gear ratio that would generate the torque necessary to rotate the gun platform. The base was connected to the firing platform by a turntable bearing, which had a central shaft that was connected to the main drive gear. A 19.1 VDC motor was fitted with an 8-tooth Technic gear, and was run through a 100:1 gear reduction. Unfortunately, despite the high gear ratio, the drive gear did not exert enough torque on the platform to rotate the system. As a result, a new design was implemented. A photo of the original base is shown in Figure 1.

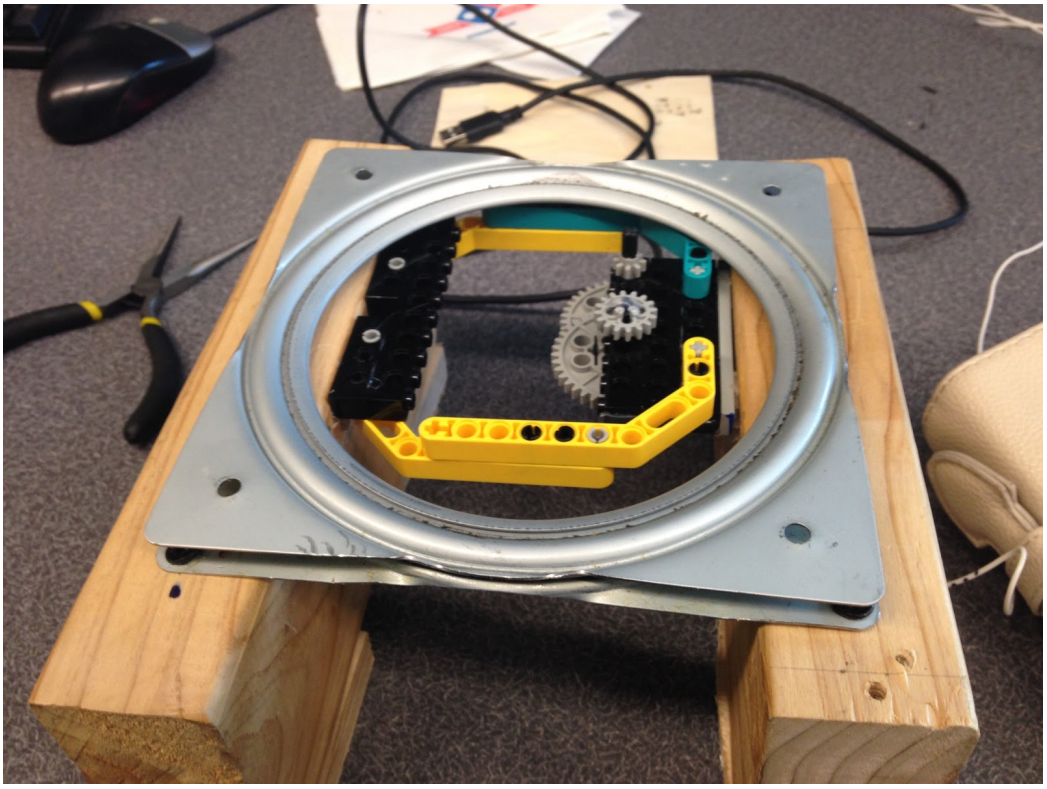


Figure 1: Photo of Original Base with Gear Train - The above photo shows the original base created for Jankbot. This base was ultimately scrapped when it did not provide sufficient torque to rotate the base.

For the second iteration of Jankbot's base, a friction drive system was implemented. Our team had seen this design successfully implemented by a few other students in our class and previous years' projects, and determined that the friction drive was the easiest and most effective solution given the time our team had. To implement this design, our team used a circular turntable from a previous quarter's robot and attached it to our platform. Our team then applied new weather-stripping along the perimeter of the turntable to increase the friction factor between the contact wheel and the base. A hole was drilled in the firing platform to allow the 19.1 VDC motor to make contact with the weather stripping. Lastly, an optical encoder was mounted to the turntable, such that the encoder base would remain stationary with respect to the firing platform, but the shaft would be free to rotate. This configuration can be seen in Figure 2 below.

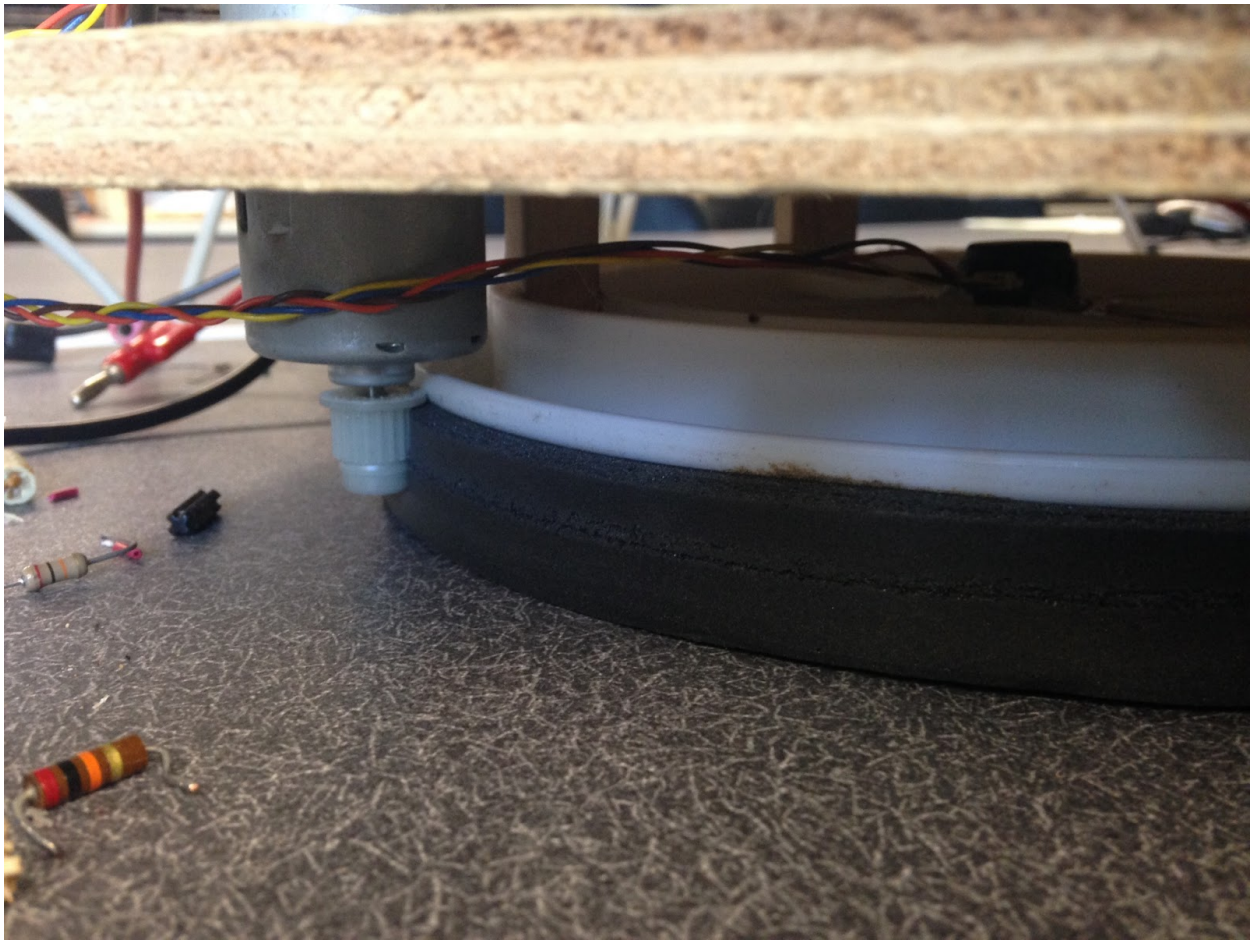


Figure 2: Friction Drive Pan Mechanism - The picture above shows the mechanism used to control the pan of the gun. The optical encoder was in the center of the Lazy Susan, and remained stationary relative to the firing platform. The motor's contact wheel exerted a torque on the base to rotate the firing platform by gripping the weather stripping on the Lazy Susan.

3.2 Tilt Axis

Similar problems to the pan axis were encountered when designing the tilt mechanism. The original idea for this design was to use a 64-tooth Lego® Technic gear fixed with JB Weld to the end of the hinge meshed with an 8-tooth pinion mounted to a 19.1 VDC motor. Theoretically, the hinge would rotate with the 64-tooth gear as it was driven by the motor. This can be seen in the concept sketches shown in Appendix A. For reasons similar to those mentioned regarding the original base design, this approach did not generate enough torque to rotate the gun platform to the desired position. The design also failed because the JB Weld could not form a strong enough bond to the hinge to withstand the torsional loads applied to it. As such, this idea was ultimately scrapped and designs for a new tilt mechanism were generated.

The next design implemented by our team was a rack and pinion jack that altered the gun's tilt. This design worked by using a 27:1 gear train to rotate a pinion which articulated a rack up or down. As the rack moved up and down, it applied a moment about the gun's hinge, which adjusted the tilt of the gun. An optical encoder was mounted to the last shaft in the gear train to

control the tilt of the gun. This was done to maximize the accuracy of the readings received from the encoder by eliminating backlash and minimizing the speed of the encoder's rotation. The full rack and pinion mechanism with gear train can be seen in Figure 3. The main issue with this design was that if the rack was pushed up too far, it would be forced out of the slot that contained it. To fix this problem, our team utilized software to enforce a limit on accepted encoder values that prevented the rack from overextending. A similar method was used to prevent the rack from being pushed lower than possible. Another issue with this design was that it did not have enough torque to push up the gun when the gun was placed all the way forward on the tilt platform. To solve this problem, our team balanced the gun's weight directly over the hinge. Washers were also taped to the gun to fine tune the balance about the hinge. This solution was able to precisely control the gun's angle, and turned out to be a very effective mechanism once it was properly balanced.

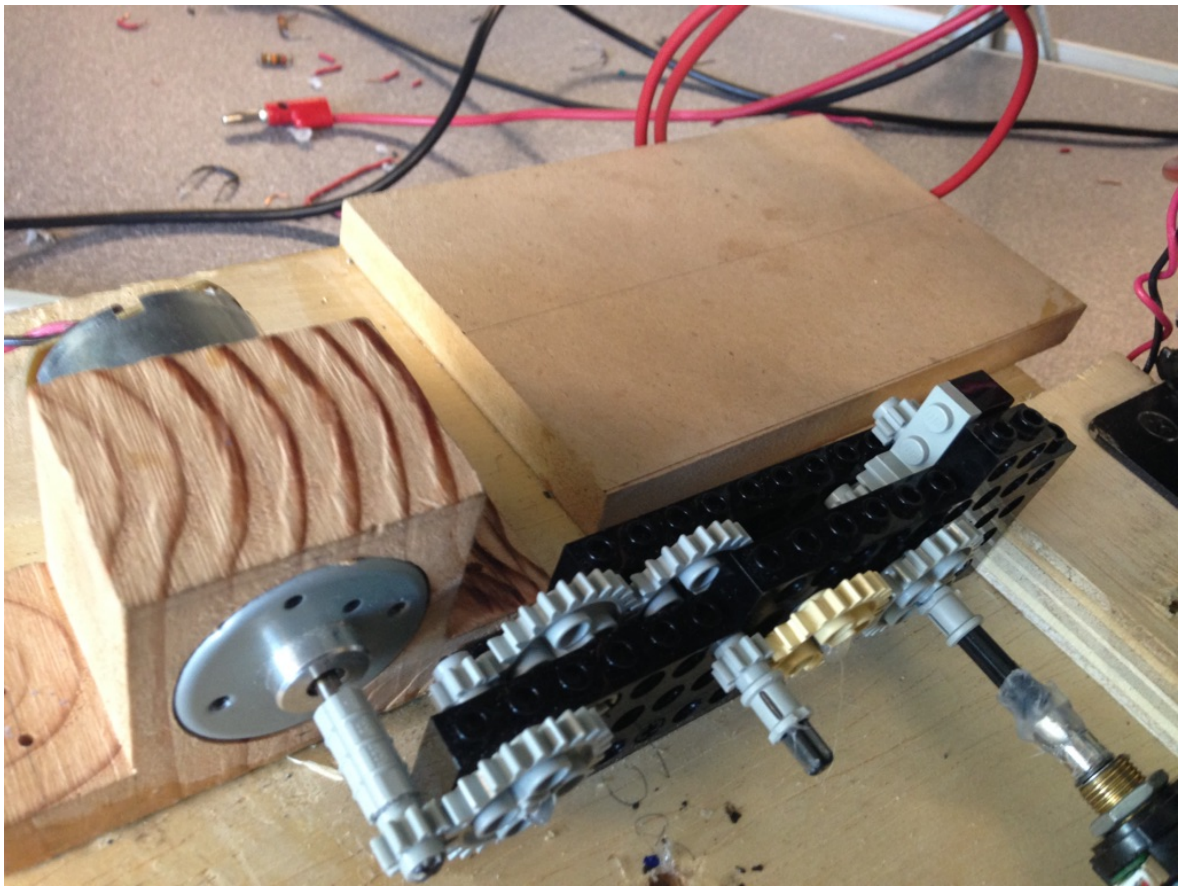


Figure 3: Rack and Pinion Tilt System - This picture shows the mechanism that controlled the gun's tilt in the final design. The optical encoder can be seen mounted to the final shaft in the gear train. This was done to maximize accuracy of the encoder readings.

3.3 Optical Sensor Design

The last major piece of hardware developed by our team was the infrared optical sensor array used to sense and focus infrared light emitted by the target. The first design for this sensor array consisted of nine phototransistors arranged in a three by three square. The sensors were wired such that each column of three transistors' collectors were wired to the same output pin on the

ME 405 board. Each row of three transistors' emitters were wired to a 20 kΩ resistor, which was connected to ground. Each row was also wired to one of three A/D converters. The original wiring diagram for this configuration can be seen in Appendix B. While this design was approved and reviewed by multiple sources including several electrical engineering students in our class and Dr. Ridgely, a very strange error was encountered when the phototransistors were wired in this manner. Whenever the collectors would receive a five volt input, the output of the sensor would exponentially decay until no measurable signal remained. While this was initially thought to be a software error, the true source of this anomaly was never discovered.

Following the first unsuccessful design of the sensor array, a new design was created according to the wiring diagram also shown in Appendix B. This design connected the collectors of five phototransistors arranged in an "X" pattern to five volts from the ME 405 board. Unlike the first design, each emitter had its own ground and A/D converter lines. This design ended up working quite well, and was only stimulated by strong sources of infrared light, such as the target used in the final test. Light was focused on the array by a magnifying glass bought from a local shop known simply as Target (it was a wondrous place of mystery and wonder). This magnifying glass was mounted approximately five and a half inches from the sensor array to properly focus the light without blind spots. The final sensor configuration and housing can be seen in Figure 6.

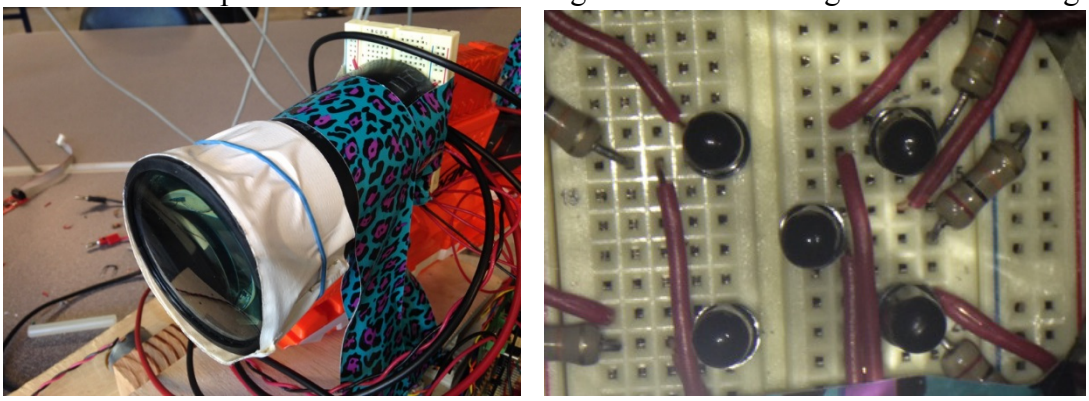


Figure 4: Phototransistor Sensor Array and Housing - The above photos show the finalized phototransistor array with its completed housing. All of the transistors were mounted on a mini-breadboard, which was directly wired to the ME 405 board.

4.0 Software Design

We designed, wrote, and debugged all the software for Jankbot. We re-used code where possible from the previous labs, including adc.cpp, encoder_driver.cpp, motor_driver.cpp, and task_encoder.cpp. Following is a list of all the files included in our project:

Table 2. Software list and brief descriptions. All *.cpp files, except for main, have a corresponding *.h file

Name	Description
adc.cpp	Class for analog to digital (A/D) converter
encoder_driver.cpp	Class for incremental optical encoder
main.cpp	Initializes all tasks and shared variables
motor_driver.cpp	This file contains a DC motor driver for the VNH3SP30 motor driver attached to the ME 405 board. This driver can control separate motors, causing them to either rotate clockwise, counterclockwise, brake, or freely rotate.
shares.h	Contains extern declarations for queues and shared variables
task_control.cpp	Performs closed loop control of two electric motors
task_encoder.cpp	Initializes two Encoder objects of from encoder_driver.cpp
task_motor.cpp	Controls the operation of a DC motor using pulse width modulation (PWM) inputs for speed set by the control loop. The motor's mode of operation is also set by the control loop.
task_position.cpp	Utilizes data from task_sensor to lock on to the light and sends position data to task_control
task_sensor.cpp	Collects data from array of phototransistors to scan for IR light.
task_trigger.cpp	This file contains the header for a task class that controls trigger of Jankbot
task_user.cpp	Contains vestigial code, mainly used to print messages to the dummy terminal

4.1 Detailed Software Descriptions

The state and task diagrams for our software can be seen in Appendix C.

4.1.1 adc.cpp

The adc.cpp file is unmodified from Lab 1 and takes inputs from adc pins 0-7 on the ME 405 board. A simple switch statement is used to set the appropriate registers based on the input channel, the case for channel 0 can be seen below:

```
case 0:
   ADMUX  &= ~(1<<MUX4)& ~(1<<MUX3)& ~(1<<MUX2)& ~(1<<MUX1)& ~(1<<MUX0);
   ADCSRA |= (1<<ADSC);
   break;
```

4.1.2 encoder_driver.cpp

The encoder_driver.cpp file was slightly modified from our Lab 4 to allow for use of two encoders. We implemented four new shared variables, p_ext_pin_A, p_ext_pin_B, p_ext_pin_C, and p_ext_pin_D. These shared variables stored the pin number each encoder was plugged into on the ME 405 board. These pins needed to be stored as shared variables so that in the ISR the correct pins could be acquired in a threadsafe manner while still retaining modularity. Because we had two different encoders we had two separate ISR subroutines, one for each encoder. As each encoder has two pins, each wired to an external interrupt pin on the ME 405 board, we aliased the ISR for pin 5 to the ISR for pin 4, and the ISR for pin 7 to the ISR for pin 6. This reduced repeated code in the file. Below is code from one of our ISR subroutines showing one of the encoder states.

```
// For Channel A and Channel B state: 00
if (!(STATE & (1<<EXT_PIN_NUMBER_A)) && !(STATE & (1<<EXT_PIN_NUMBER_B)))
{
	// If previous state was 01, increment ENCODER_COUNT in positive
	direction
	if (!(STATE_OLD & (1<<EXT_PIN_NUMBER_A)) && (STATE_OLD & (1
	<<EXT_PIN_NUMBER_B)))
	{
		ENCODER_COUNT++;
	}

	// If previous state was 10, increment ENCODER_COUNT in negative
	direction
	else if ((STATE_OLD & (1<<EXT_PIN_NUMBER_A)) && !(STATE_OLD & (1
	<<EXT_PIN_NUMBER_B)))
	{
		ENCODER_COUNT--;
	}

	// If previous state was neither 01 or 10, increment ERROR_COUNT
	else
	{
		ERROR_COUNT++;
	}
}
```

4.1.3 motor_driver.cpp

The motor_driver.cpp file was not modified from Lab 3. Our team only modified the corresponding Output Compare Register as specified in the constructor.

4.1.4 task_control.cpp

The task_control.cpp file was based on our original Lab 5 code and modified to accommodate two different motors. We utilized the shared variables p_position_1 and p_position_2 to set the reference position for the control loop. The control loop subtracts the current position from the reference position to obtain the error for each respective motor. Then the error is multiplied by the closed loop proportional control gain. The error is also summed with the motor's previous error and multiplied by the closed loop integral control gain. We have the capability to

implement proportional, integral, and derivative control. However for this specific control loop we only implement proportional and integral gains. Then after the error is multiplied by the closed loop gains, we implemented a tolerance for which the motor enters a braked state. It is during this braked state that the task changes the shared variables p_pos_done_1 and p_pos_done_2 to true, signifying that the system has reached the desired position. Additionally we implemented a dead zone for which the controller outputs a minimum speed to overcome the stiction of the motor. There was also a speed cap that was implemented to ensure smoother operation of the motor. Finally, for the tilt control we implemented a limit for which the controller will brake the motor before the rack will fall out of the gear train.

We encountered a strange bug where if we did not print the reference position, actual position, and speed_out then the motors would not spin. We were unable to diagnose the cause of this bug in the time we had left, but we suspected that this may have been a timing issue. However after changing the timing we were unable to rectify the problem.

4.1.5 task_encoder.cpp

The encoder task simply instantiates two objects of the Encoder class defined in motor_driver.cpp, one for each encoder in the system. p_encoder_1 corresponds to encoder measuring the gun tilt and p_encoder_2 corresponds to the encoder measuring base rotation.

4.1.6 task_motor.cpp

The motor task instantiates the two objects of the Motor class defined in motor_driver.cpp, one for each motor in the system. p_motor_1 corresponds to the motor driving the gun angle and p_motor_2 corresponds to the motor driving the base rotation. Then the task initializes the pulse width modulation (PWM) for fast 8-bit PWM and the prescaler to 64. Then the task calls the relevant methods from motor_driver.cpp for each motor based on the state of the shared variable p_mode. The code for the switch statement for the mode is shown below:

```
switch(mode)
{
	case(brake_1):
		p_motor_1->brake();
		break;

	case(free_1):
		p_motor_1->freewheel();
		break;

	case(power_1):
		p_motor_1->set_power(speed_1);
		break;
```

4.1.7 task_position.cpp

The position task has three states, the first state simply rotates the gun approximately 160 degrees from the start position to face the target area. Then it waits for the shared variables p_pos_done_1 and p_pos_done_2 variable to be set to true by task_control.cpp before transitioning to the next state.

In the next state the position task follows a search pattern to attempt to locate the light source. The pattern begins at the lower right hand corner of the target area and sweeps left to the base_l_limit. If the light source is not seen then the gun angle is increased and then swept right to the base_r_limit. If the light source is not seen then the gun angle is then increase and the gun is swept left, and thus the pattern continues until the light source is identified. Once the light source is seen, as determined by any of the phototransistors reading above the threshold variable, then the task transitions to the third state where the position task locks onto the target using feedback from the sensor task. Below is the state transition to the lock on state:

```
// If any sensor reads above the threshold transition to state 2
if((high_left >= threshold) || (high_right >= threshold) ||
      (center >= threshold) || (low_left >= threshold) ||
      (low_right >= threshold))
{
      transition_to (2);
}
```

In the lock on state the position task attempts to move the gun so that the center phototransistor is reading a maximum voltage over all the other phototransistors. This is accomplished by comparing the upper and lower sensors to the center and changing the gun tilt accordingly as well as comparing the left and right sensors to the center and changing the base position accordingly. Once the gun is position such that the center phototransistor A/D reading is within a ± tolerance of all the other phototransistor readings then the shared variable fire_at_will is set to true so that the trigger may be pulled in task_trigger.

```
if ((center > (low_right - tol)) && (center > (high_right - tol))
      && (center > (low_left - tol)) && (center > (low_right - tol))
      && (center < (low_right + tol)) && (center < (high_right + tol))
      && (center < (low_left - tol)) && (center < (low_right + tol)))
{
      fire_at_will->put(true);
}
```

4.1.8 task_sensor.cpp

The sensor task simply read the five phototransistors using the adc method read_oversampled. We took an oversampling of 4 for each phototransistor in order to average out some of the noise from each reading. Each reading was then placed into a corresponding shared variable: p_high_left, p_high_right, p_center, p_low_left, p_low_right.

4.1.9 task_trigger.cpp

The trigger task runs the servo when the shared variable fire_at_will is set to true for 100 runs of the task, so for 5 seconds. After 5 seconds the servo is set back to its initial position and fire_at_will is set back to false.

4.1.10 task_user.cpp

The primary use for the user task is the enabling of printing to the dummy terminal, enabling the p_ser_queue to be utilized in the other tasks. It also contains vestigial code for debugging purposes.

5.0 Results

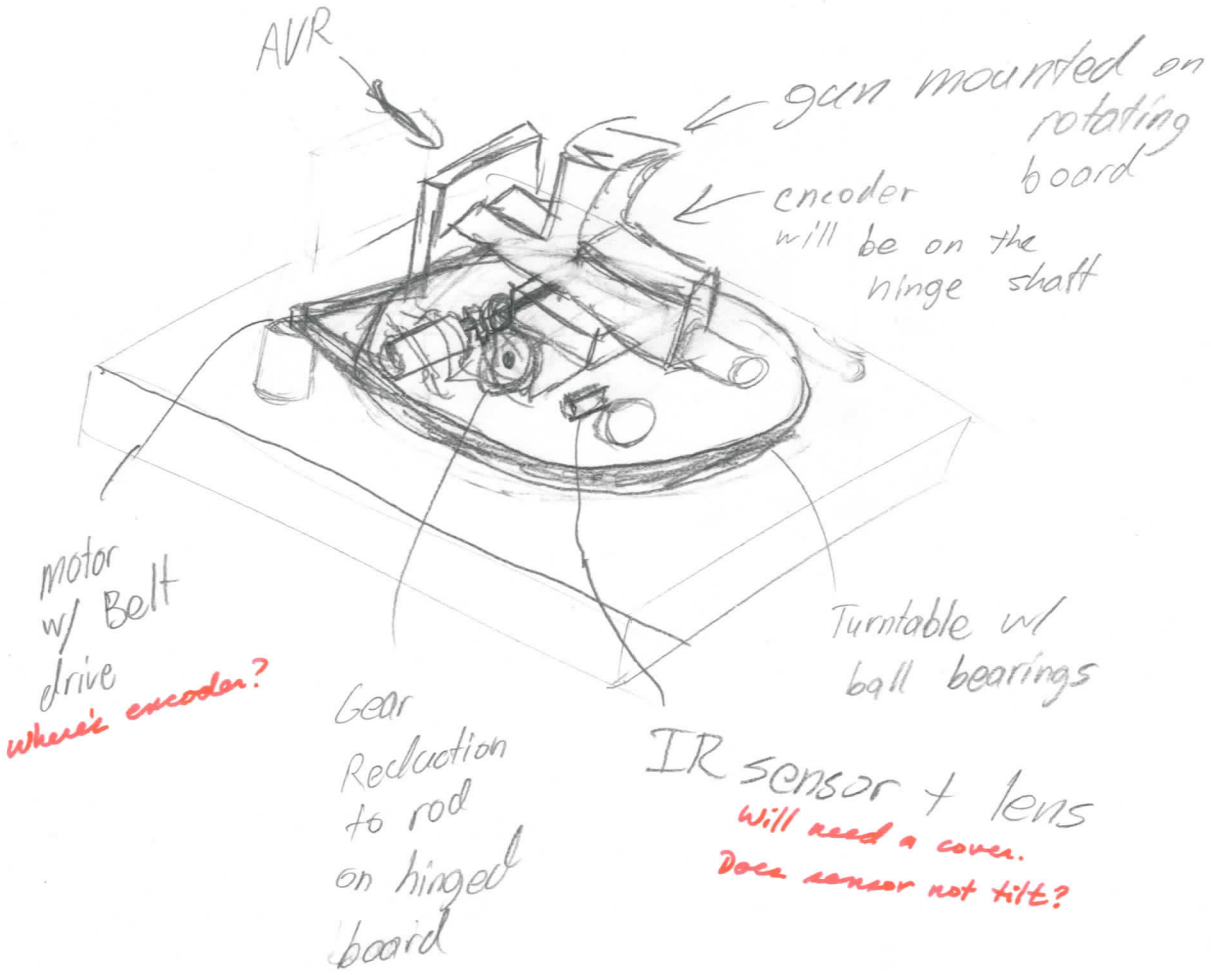
After all of our hard work, our team's project ended up not coming together in time. Jankbot was able to pan, tilt, and receive viable readings from the IR sensors, but our team was unable to successfully implement these features in a closed control loop. Our team encountered many setbacks along the way that ended up costing us most of our time. These setbacks included the axes' control hardware, optical sensor wiring, and strange errors that inhibited cooperative multitasking. We were able to show that the control axes worked by setting up code to move Jankbot to certain positions in the x-axis and y-axis, and we were also able to show that the infrared sensor array worked by printing the readings obtained from the phototransistors when the sensor was exposed to infrared light and then to ambient conditions. These test results are shown in Appendix C.

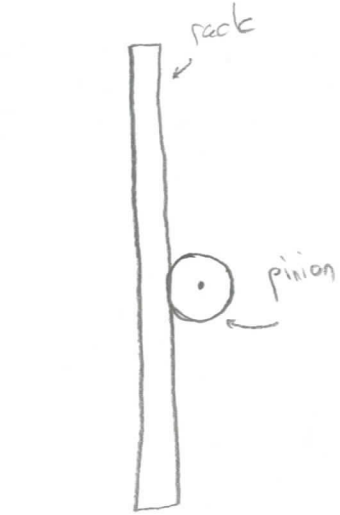
As a learning experience, this project was very successful. Our team learned how to properly wire phototransistors in an array to measure infrared light and output the readings from these sensors to an A/D converter. We also developed a sense for the importance that timing and priority have when using many tasks in a given application. On the note of timing, our group suspects that timing issues were the prime reasons that the control loop was failing. Lastly, our team also accidentally learned a lot about different types of glue and their applications. For example, using hot glue to hold a plastic gear to a metal hinge end that is receiving bending forces is not the proper use for that glue. Our use or misuse of various types of glue lead to some very interesting, albeit sticky, situations.

If we were to do this project again, there are a few things our team would do differently. First, we would start constructing the design earlier. For this project, we started constructing axes hardware only a week before we planned on having it implemented. Our team met many unexpected challenges throughout the construction process and had to rethink and change our original designs, which increased the amount of time required for construction. We would also have spent less time trying to solve the decay issue with the nine-phototransistor array and instead would have focused on creating a new, working arrangement of the phototransistors. Overall, the greatest takeaway from this project was that it is often more worthwhile to come up with a new design then to try and make a broken design functional. If we had not been so attached to our initial ideas, we would have moved through some of the more difficult and time-wasting technical issues much faster. This would have given us more time to focus on our software design. While our team was unable to make Jankbot fully autonomous, it is fully possible that if we could have made a working system if we had made the changes detailed above. This experience gave us all valuable insight that we will use in projects to come.

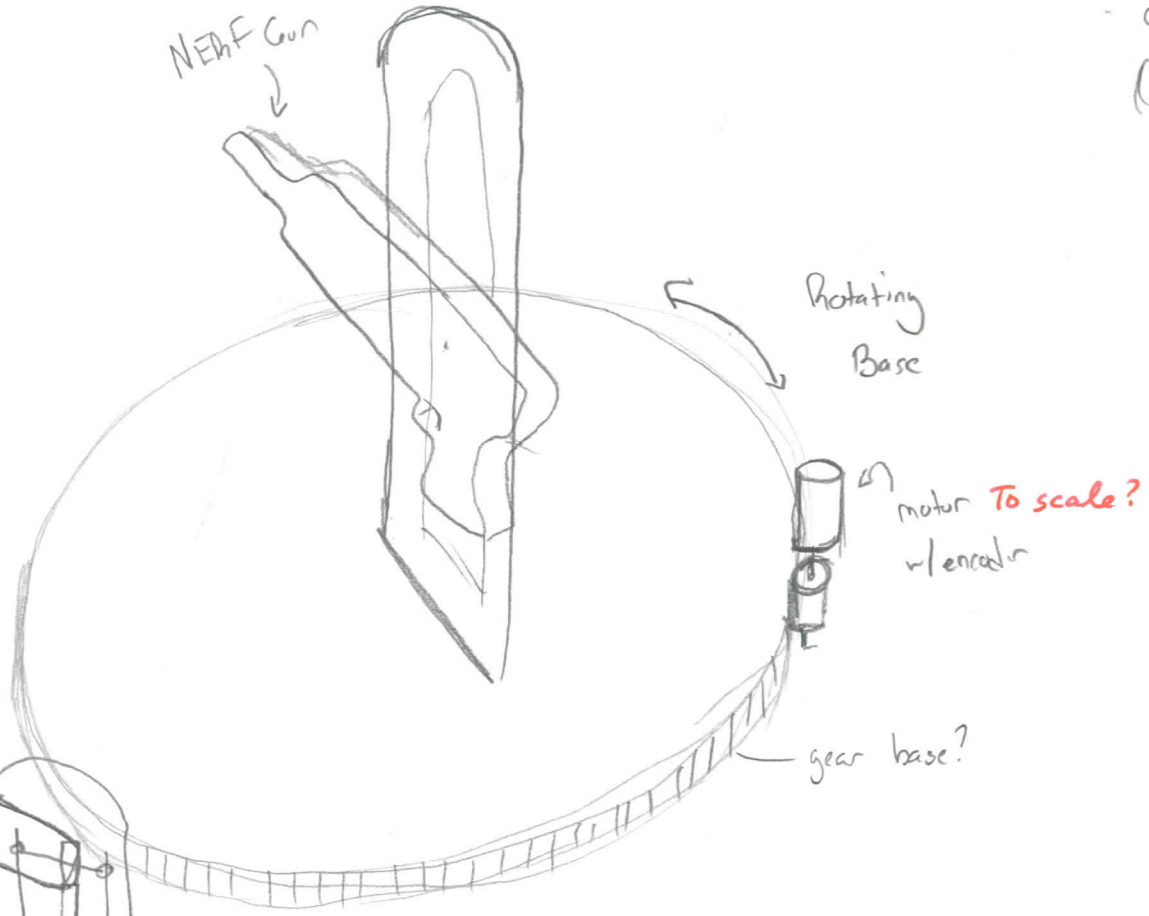
6.0 Appendices

Appendix A: Design Drawings

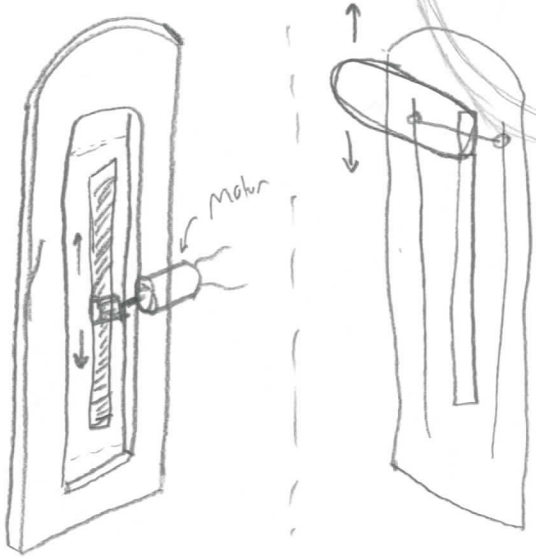


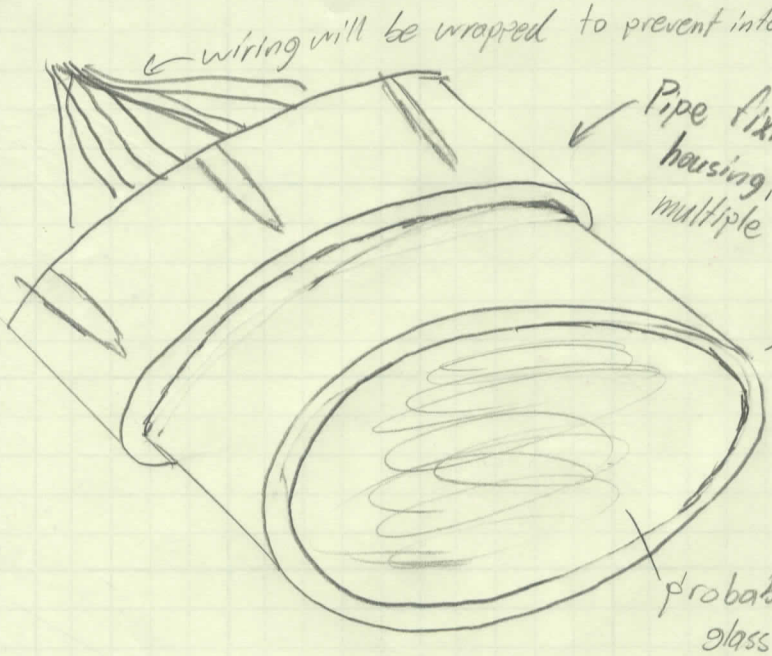


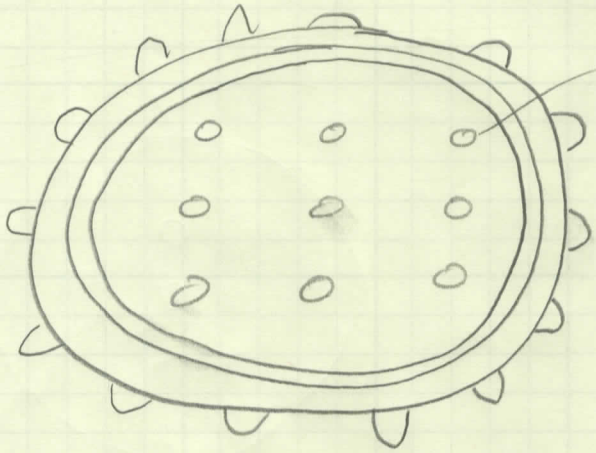
Lazy susan base
- gear base
or. belt drive
(depends on reduction needed)



Linear Drive





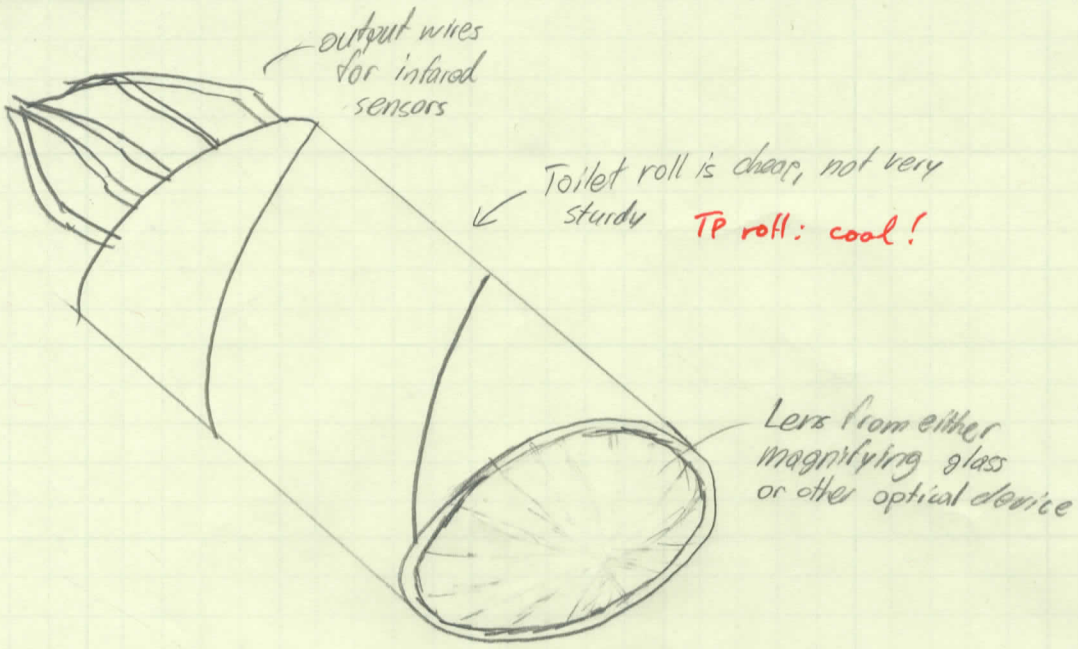


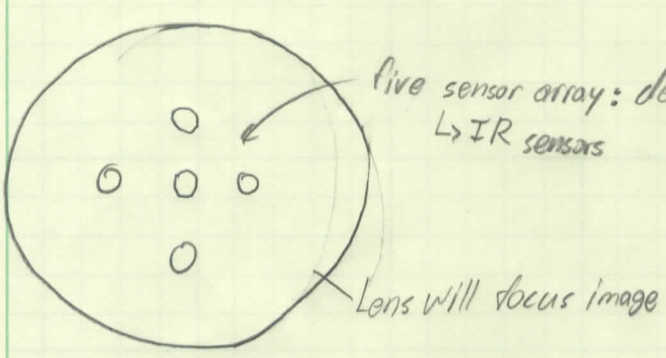
bigger housing allows for more sensors. May be more precise ok but probably not needed

Pros: Housing is more rigid / durable, can still be mounted to gun, more sensors, a grid of sensors vs. a cross.

Cons: Heavier / may impede y-axis adjustment, not as easy to cut to length.

Toilet Roll Idea

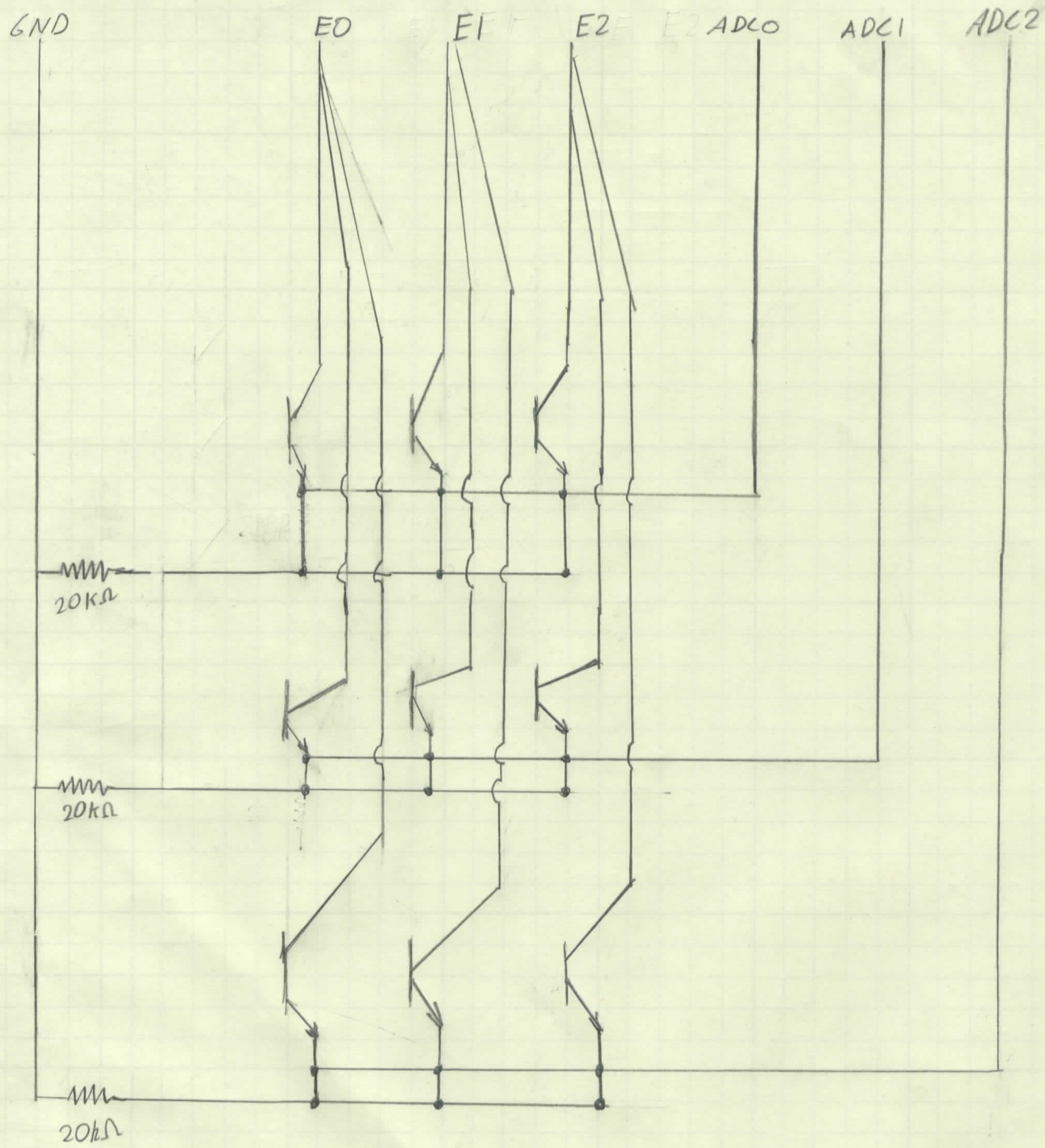




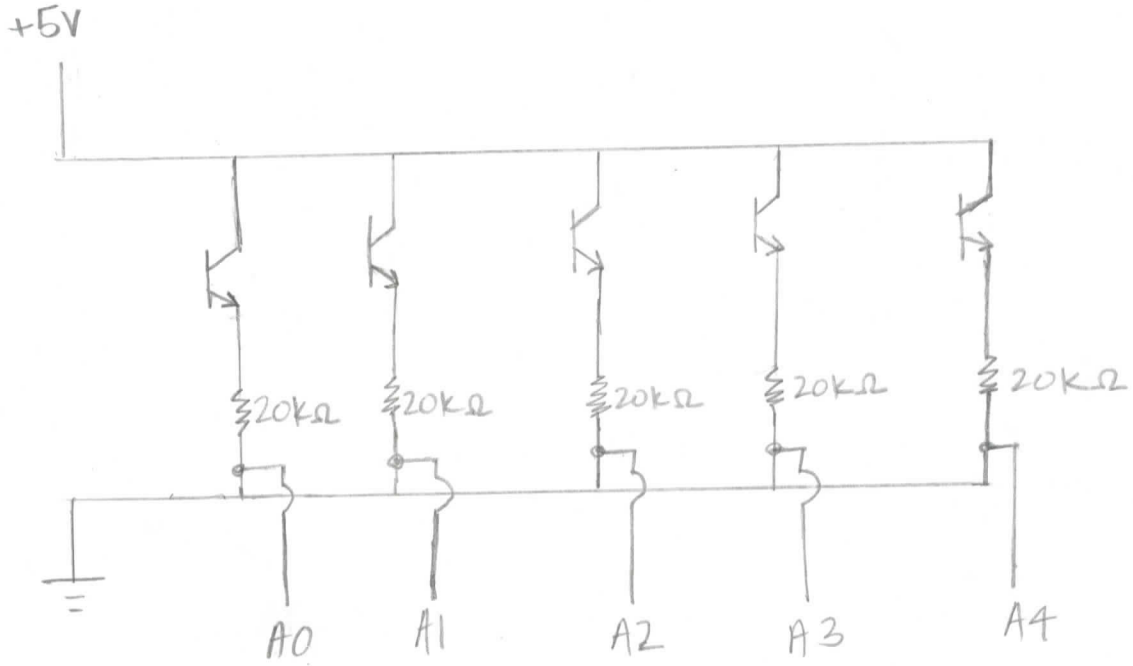
Pros: Cheap, readily available materials, easily mountable to the gun, can be cut to focal length.

Cons: Weak/flimsy shell, needs reinforcement tube has small diameter, which limits sensor spacing.

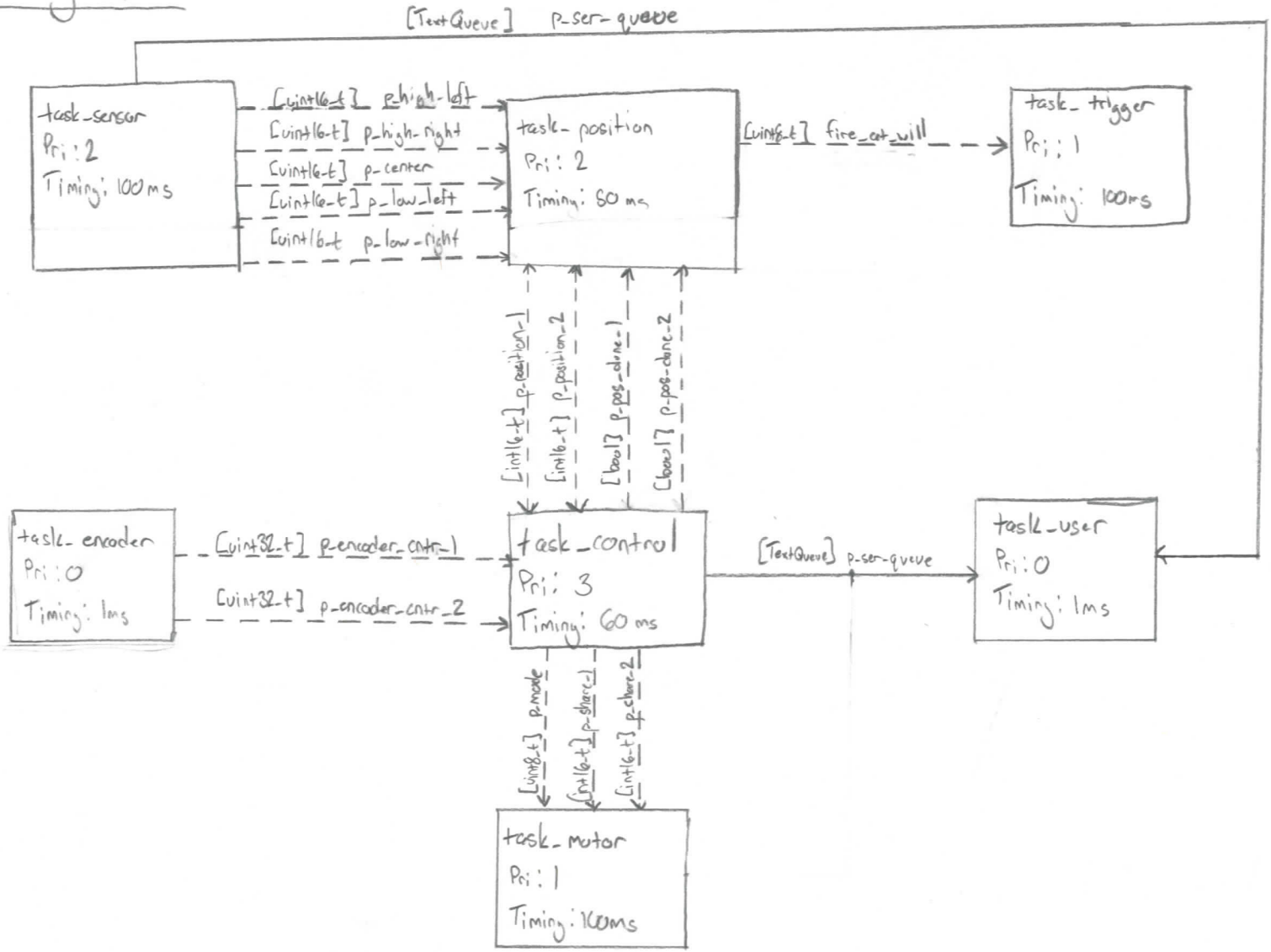
Appendix B: Wiring Diagram



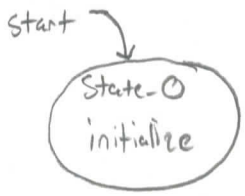
Final Sensor Schematic



Task Diagram

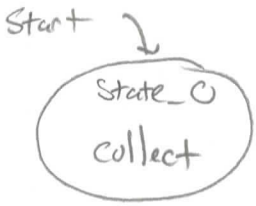


State Diagram : task_encoder

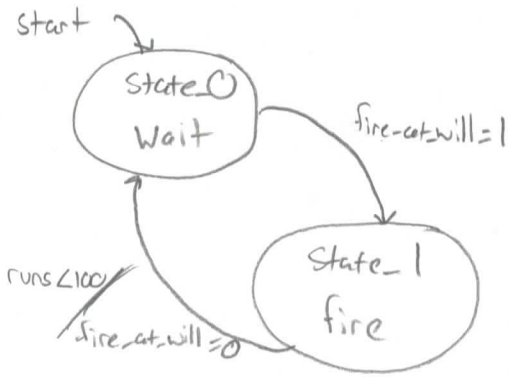


State Diagram : task_motor

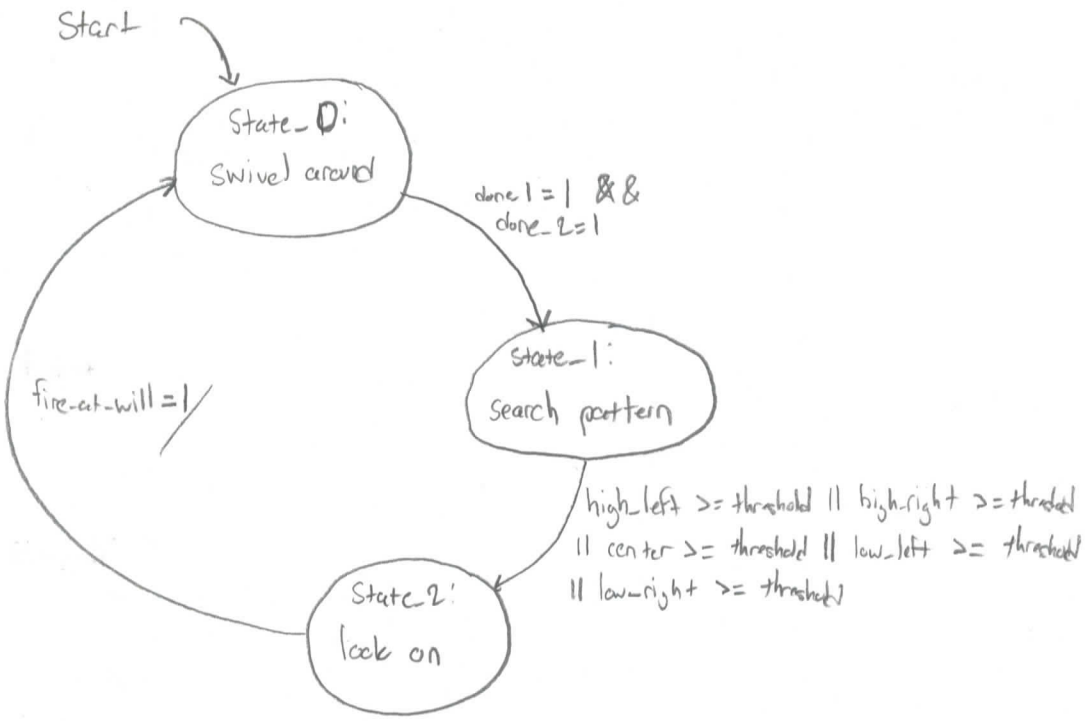
State Diagram : task_sensor



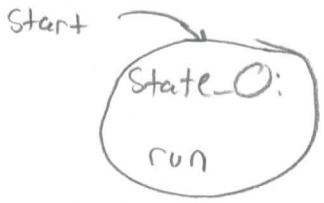
State Diagram : task_trigger



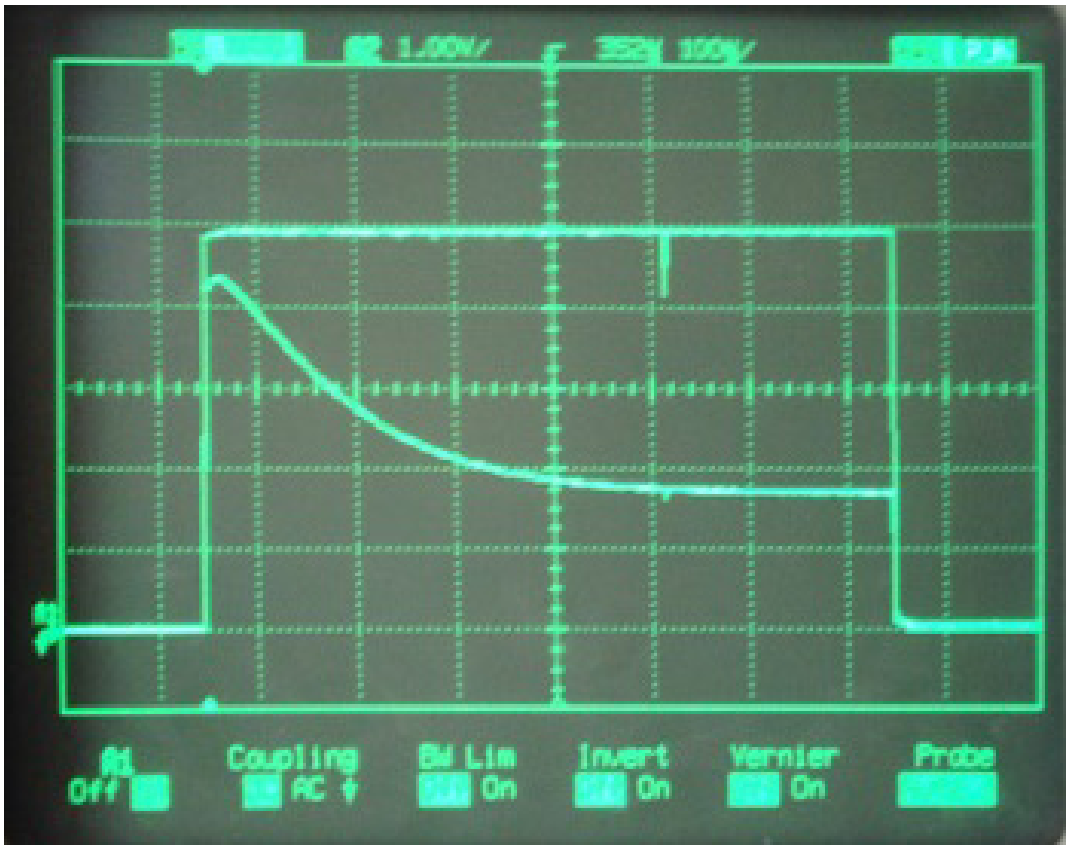
State Diagram : task_position



State Diagram : task_control



Appendix D: Multiplex Sensor Readings



Final Project: Learn By Dueling

Chris Lutze, Jen Hawthorne, Tristan Thompson

Version 1

Hierarchical Index

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

File Index

File List

Here is a list of all documented files with brief descriptions:

Class Documentation

adc Class Reference

This class **will** run the A/D converter on an AVR processor.

`#include <adc.h>`

Public Member Functions

adc (emstream *=NULL)

This constructor sets up an A/D converter.

uint16_t **read_once** (uint8_t)

This method takes one A/D reading from the given channel and returns it.

uint16_t **read_oversampled** (uint8_t, uint8_t)

This method averages multiple readings from the A/D conversion and returns the average.

Protected Attributes

emstream * **ptr_to_serial**

The ADC class uses this pointer to the serial port to say hello.

Detailed Description

This class will run the A/D converter on an AVR processor.

The class contains a protected pointer to the serial port for outputs. Public functions read_once and read_oversampled perform A/D conversions and emstream prints the A/D conversion value for each channel and the ADSCRA, ADMUX registers.

Definition at line 48 of file adc.h.

Constructor & Destructor Documentation

adc::adc (emstream * *p_serial_port* = `NULL`)

This constructor sets up an A/D converter.

The A/D is made ready so that when a method such as **`read_once()`** is called, correct A/D conversions can be performed. The ADC Control and Status Register is set such that the reference voltage source is from AVCC with external Capacitor at AREF pin and the clock prescaler is at a division factor of 32.

Parameters:

p_serial_port	A pointer to the serial port which writes debugging info.

Definition at line 42 of file adc.cpp.

References ptr_to_serial.

Member Function Documentation

uint16_t adc::read_once (uint8_t *ch*)

This method takes one A/D reading from the given channel and returns it.

This function performs a single A/D conversion based on the paramenter ch. The selected channel input is read and the ADMUX and ADCSRA registers are set accordingly. The A/D conversion is initiated and the code waits until the ADSC bit returns to 0, signifying the conversion is complete. Finally, the A/D conversion result is stored and the value returned.

Parameters:

ch	The A/D channel which is being read must be from 0 to 7

Returns:

The result of the A/D conversion

Definition at line 65 of file adc.cpp.

Referenced by read_oversampled().

uint16_t adc::read_oversampled (uint8_t *channel*, uint8_t *samples*)

This method averages multiple readings from the A/D conversion and returns the average.

This function takes a set number of samples from the read_once function and averages them together to reduce the effects of noise.

Parameters:

channel	The A/D channel which is being read must be from 0 to 7.
samples	Number of samples to be averaged.

Returns:

The result of the averaged A/D conversion

Definition at line 140 of file adc.cpp.

References read_once().

Referenced by task_scan::run().

The documentation for this class was generated from the following files:

adc.h
adc.cpp

Encoder Class Reference

This class will read an encoder connected to the AVR processor by any of the external input pins 4 through 7.

`#include <encoder_driver.h>`

Public Member Functions

Encoder (emstream *=NULL, volatile uint8_t *i_port=NULL, volatile uint8_t *i_ddr=NULL, volatile uint8_t *i_port_in=NULL, uint8_t i_pin_0_a=0, uint8_t i_pin_1_a=0, uint8_t e_pin_a=0, uint8_t i_pin_0_b=0, uint8_t i_pin_1_b=0, uint8_t e_pin_b=0)

This constructor sets up an instance of the class encoder_driver.

uint16_t **error_count** (uint8_t enc_num)

Displays the current error count.

void **clear_count** (uint8_t enc_num)

Sets the encoder count to 0.

void **view_count** (uint8_t enc_num)

Displays the current encoder count.

void **set_count** (uint8_t enc_num, uint32_t NEW_COUNT)

Sets the encoder count to a specific input.

Protected Attributes

emstream * **ptr_to_serial**

*The **Encoder** class uses this pointer to the serial port to say hello.*

volatile uint8_t * **INTERRUPT_PORT**
volatile uint8_t * **INTERRUPT_DDR**
uint8_t **INTERRUPT_PIN_0_A**
uint8_t **INTERRUPT_PIN_1_A**
uint8_t **INTERRUPT_PIN_0_B**
uint8_t **INTERRUPT_PIN_1_B**
uint8_t **EXT_PIN_NUMBER_A**
uint8_t **EXT_PIN_NUMBER_B**
uint32_t **ENCODER_COUNT**
uint32_t **ERROR_COUNT**
uint8_t **STATE**
uint8_t **STATE_OLD**

Detailed Description

This class will read an encoder connected to the AVR processor by any of the external input pins 4 through 7.

The class contains a protected pointer to the serial port for outputs. It also contains pointers for encoder PORTs, DDRs, and pin addresses. Public function error_count displays the number of errors accumulated. Public function clear_count zeros the encoder count. Public function

view_count displays the current encoder count. Public function set_count sets the encoder count according to a user input.

Definition at line 53 of file encoder_driver.h.

Constructor & Destructor Documentation

Encoder::Encoder (emstream * *p_serial_port* = `NULL`, volatile uint8_t * *i_port* = `NULL`, volatile uint8_t * *i_ddr* = `NULL`, volatile uint8_t * *i_port_in* = `NULL`, uint8_t *i_pin_0_a* = `0`, uint8_t *i_pin_1_a* = `0`, uint8_t e*_pin_a* = `0`, uint8_t *i_pin_0_b* = `0`, uint8_t *i_pin_1_b* = `0`, uint8_t *e_pin_b* = `0`)

This constructor sets up an instance of the class encoder_driver.

The encoder is made ready so that when a method such as **`view_count()`** is called, the corresponding registers can be correctly set. The constructor contains a protected pointer to the serial port for outputs. The constructor contains a protected pointer to the serial port for outputs. It also sets pointers for encoder PORTs, DDRs, and pin addresses. It generates an interrupt to run on any logic change on the external interrupt input pins.

Parameters:

p_serial_port	A pointer to the serial port which writes debugging info.
i_port	A pointer to the corresponding PORT for the encoder.
i_ddr	A pointer to the corresponding DDR for the encoder.
i_pin_0_a	The pin number for EICRB register bit 0 of external pin A the encoder is wired to
i_pin_1_a	The pin number for EICRB register bit 1 of external pin A the encoder is wired to
e_pin_a	The pin number for the external input pin A the encoder is wired to.
i_pin_0_b	A pointer to the EICRB register bit 0 of the external pin B the encoder is wired to
i_pin_1_b	The pin number for EICRB register bit 0 of external pin B the encoder is wired to
e_pin_b	The pin number for the external input pin A the encoder is wired to.

Definition at line 60 of file encoder_driver.cpp.

References ptr_to_serial.

Member Function Documentation

void Encoder::clear_count (uint8_t *enc_num*)

Sets the encoder count to 0.

Sets the share variable p_encoder_cntr to 0

Definition at line 140 of file encoder_driver.cpp.

Referenced by task_user::run().

uint16_t Encoder::error_count (uint8_t *enc_num*)

Displays the current error count.

By using the shared variable p_error_cntr the error count is pulled and displayed through a debug message sent to the serial port.

Definition at line 118 of file encoder_driver.cpp.

void Encoder::set_count (uint8_t *enc_num*, uint32_t *NEW_COUNT*)

Sets the encoder count to a specific input.

Sets the shared variable p_encoder_cntr to the NEW_COUNT

Parameters:

NEW_COUNT	A uint32_t variable containing the new encoder count number

Definition at line 182 of file encoder_driver.cpp.

Referenced by task_user::run().

void Encoder::view_count (uint8_t *enc_num*)

Displays the current encoder count.

Collects data from the shared variable p_encoder_cntr and displays it through a debug message sent to the serial port.

Definition at line 160 of file encoder_driver.cpp.

The documentation for this class was generated from the following files:

encoder_driver.h
encoder_driver.cpp

Motor Class Reference

This class will operate the VNH3SP30 motor driver on an AVR processor.

`#include <motor_driver.h>`

Public Member Functions

Motor (emstream *=NULL, volatile uint8_t *a_port=NULL, volatile uint8_t *a_d=NULL, uint8_t a_pin=0, volatile uint8_t *b_port=NULL, volatile uint8_t *b_d=NULL, uint8_t b_pin=0, volatile uint8_t *diag_po=NULL, volatile uint8_t *diag_d=NULL, uint8_t diag_pi=0, volatile uint8_t *pwm_po=NULL, volatile uint8_t *pwm_d=NULL, uint8_t pwm_pi=0, volatile uint16_t *pwm_oc=NULL)

This constructor sets up a motor driver to work with the VNH3SP30 motor driver.

void **set_power** (int16_t speed)

Takes a 16 bit signed input and sets the motor torque.

void **freewheel** (void)

This method toggles motor freewheeling.

void **brake** (void)

Sets the motor to a braked state.

Protected Attributes

emstream * **ptr_to_serial**

*The **Motor** class uses this pointer to the serial port to say hello.*

volatile uint8_t * **INa_PORT**
volatile uint8_t * **INa_DDR**
uint8_t **INa_PIN**
volatile uint8_t * **INb_PORT**
volatile uint8_t * **INb_DDR**
uint8_t **INb_PIN**
volatile uint8_t * **DIAG_PORT**
volatile uint8_t * **DIAG_DDR**
uint8_t **DIAG_PIN**
volatile uint8_t * **PWM_PORT**
volatile uint8_t * **PWM_DDR**
uint8_t **PWM_PIN**
volatile uint16_t * **PWM_OCR**

Detailed Description

This class will operate the VNH3SP30 motor driver on an AVR processor.

The class contains a protected pointer to the serial port for outputs. It also contains pointers for motor driver PORTs, DDRs, and pin addresses. Public function set_power controls the motor speed based on a signed 16-bit input. Public function brake grounds both the motor leads so it will not spin. Public function freewheel sets the motor speed to zero, allowing it to rotate freely.

Definition at line 53 of file motor_driver.h.

Constructor & Destructor Documentation

Motor::Motor (emstream * *p_serial_port* = `NULL`, volatile uint8_t * *a_port* = `NULL`, volatile uint8_t * *a_d* = `NULL`, uint8_t *a_pin* = `0`, volatile uint8_t * *b_port* = `NULL`, volatile uint8_t * *b_d* = `NULL`, uint8_t *b_pin* = `0`, volatile uint8_t * *diag_po* = `NULL`, volatile uint8_t * *diag_d* = `NULL`, uint8_t *diag_pi* = `0`, volatile uint8_t * *pwm_po* = `NULL`, volatile uint8_t * *pwm_d* = `NULL`, uint8_t *pwm_pi* = `0`, volatile uint16_t * *pwm_oc* = `NULL`)

This constructor sets up a motor driver to work with the VNH3SP30 motor driver.

The motor driver is made ready so that when a method such as **`set_power()`** is called, the corresponding registers on the VNH3SP30 can be correctly set. The constructor contains a protected pointer to the serial port for outputs. It also contains pointers for motor driver PORTs, DDRs, and pin addresses. Public function set_power controls the motor speed based on a signed 16-bit input. Public function brake grounds both the motor leads so it will not spin. Public function freewheel sets the motor speed to zero, allowing it to rotate freely.

Parameters:

p_serial_port	A pointer to the serial port which writes debugging info.
a_port	A pointer to the port corresponding to VNH3SP30 pin INa
a_d	A pointer to the Data Direction Register for VNH3SP30 pin INa
a_pin	An integer for the pin/bit location corresponding to VNH3SP30 pin INa
b_port	A pointer to the port corresponding to VNH3SP30 pin INb
b_d	A pointer to the Data Direction Register for VNH3SP30 pin INb
b_pin	An integer for the pin/bit location corresponding to VNH3SP30 pin INb
diag_po	A pointer to the port corresponding to VNH3SP30 diagnostic pin: DIAGA/B
diag_d	A pointer to the Data Direction Register for VNH3SP30 diagnostic pin: DIAGA/B
diag_pi	An integer for the pin/bit location corresponding to VNH3SP30 diagnostic pin: DIAGA/B
pwm_po	A pointer to the port corresponding to VNH3SP30 PWM pin
pwm_d	A pointer to the Data Direction Register for VNH3SP30 pin INb
pwm_pi	An integer for the pin/bit location corresponding to VNH3SP30 pin INa
pwm_oc	A pointer to the Output Compare Register for the PWM

Definition at line 66 of file motor_driver.cpp.

References ptr_to_serial.

Member Function Documentation

void Motor::brake (void)

Sets the motor to a braked state.

Grounds both the INa and INb pins so that the motor is braked. Sets the PWM for maximum braking power.

Definition at line 153 of file motor_driver.cpp.

Referenced by task_motor::run().

void Motor::freewheel (void)

This method toggles motor freewheeling.

By setting the PWM Output Compare Register to zero, the motor will spin freely.

Definition at line 142 of file motor_driver.cpp.

Referenced by task_motor::run().

void Motor::set_power (int16_t *speed*)

Takes a 16 bit signed input and sets the motor torque.

A positive input speed will spin the motor clockwise by setting INa to high and set the PWM Output Compare Register to the input speed. A negative input speed will spin the motor counterclockwise by setting INb to high and set the PWM Output Compare Register to the input speed.

Parameters:

speed	A value used to set the PWM duty cycle

Definition at line 116 of file motor_driver.cpp.

Referenced by task_motor::run().

The documentation for this class was generated from the following files:

motor_driver.h
motor_driver.cpp

task_control Class Reference

This task controls and reads a motor with an encoder.

`#include <task_control.h>`

Public Member Functions

task_control (const char *, unsigned portBASE_TYPE, size_t, emstream *)
void **run** (void)

Protected Attributes

int32_t **current_pos_1**
int32_t **current_pos_2**
int32_t **ref_pos_1**
int32_t **ref_pos_2**
const double **KP_1** = .5
const double **KI_1** = .05
const double **KP_2** = 1
const double **KI_2** = .01
int8_t **KD**
int32_t **error_old_1**
int32_t **error_old_2**
int32_t **error_1**
int32_t **error_2**
double **KP_out_1**
double **KP_out_2**
double **KI_out_1**
double **KI_out_2**
double **KD_out_1**
double **KD_out_2**
double **speed_out_1**
double **speed_out_2**
int8_t **dead_zone**
uint16_t **hinge_limit**
uint8_t **count**

Detailed Description

This task controls and reads a motor with an encoder.

The encoder is read and controller is run using a driver in files **`encoder_driver.h`** and **`encoder_driver.cpp`** .

Definition at line 56 of file task_control.h.

Constructor & Destructor Documentation

task_control::task_control (const char * *a_name*, unsigned portBASE_TYPE *a_priority*, size_t *a_stack_size*, emstream * *p_ser_dev*)

This constructor creates a task which reads input from an encoder and controls the encoder using input from **`task_user`** . The main job of this constructor is to call the constructor of parent class (`frt_task`).

Parameters:

a_name	A character string which will be the name of this task
a_priority	The priority at which this task will initially run (default: 0)
a_stack_size	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)
p_ser_dev	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)

Definition at line 50 of file task_control.cpp.

Member Function Documentation

void task_control::run (void)

This method is called once by the RTOS scheduler. It constructs the encoder to run using external interrupts. Each time around the for (;;) loop, the encoder is updated with the latest shared variables from the motor and/or **task_user**.

Definition at line 66 of file task_control.cpp.

References p_print_ser_queue.

The documentation for this class was generated from the following files:

task_control.h
task_control.cpp

task_encoder Class Reference

This task controls and reads an encoder.

`#include <task_encoder.h>`

Public Member Functions

task_encoder (const char *, unsigned portBASE_TYPE, size_t, emstream *)
void **run** (void)

Detailed Description

This task controls and reads an encoder.

The encoder readed and controler is run using a driver in files **`encoder_driver.h`** and **`encoder_driver.cpp`** .

Definition at line 60 of file task_encoder.h.

Constructor & Destructor Documentation

task_encoder::task_encoder (const char * *a_name*, unsigned portBASE_TYPE *a_priority*, size_t *a_stack_size*, emstream * *p_ser_dev*)

This constructor creates a task which reads input from an encoder and controls the encoder using input from **`task_user`** . The main job of this constructor is to call the constructor of parent class (`frt_task`).

Parameters:

a_name	A character string which will be the name of this task
a_priority	The priority at which this task will initially run (default: 0)
a_stack_size	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)
p_ser_dev	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)

Definition at line 42 of file task_encoder.cpp.

Member Function Documentation

void task_encoder::run (void)

This method is called once by the RTOS scheduler. It constructs the encoder to run using external interrupts. Each time around the for (;;) loop, the encoder is updated with the latest shared variables from the motor, control loop, and/or **task_user**.

Definition at line 57 of file task_encoder.cpp.

The documentation for this class was generated from the following files:

task_encoder.h
task_encoder.cpp

task_motor Class Reference

This task controls two separate motors using a motor driver and shared variables set in **task_user**.

`#include <task_motor.h>`

Public Member Functions

task_motor (const char *, unsigned portBASE_TYPE, size_t, emstream *)
void **run** (void)

Protected Attributes

uint8_t **mode**
int16_t **speed_1**
int16_t **speed_2**

Detailed Description

This task controls two separate motors using a motor driver and shared variables set in **task_user**.

The motor controller is run using a driver in files **`motor_driver.h`** and **`motor_driver.cpp`** . Code in this task sets up a timer/counter in PWM mode and controls the motors' modes and run speeds.

Definition at line 62 of file task_motor.h.

Constructor & Destructor Documentation

task_motor::task_motor (const char * *a_name*, unsigned portBASE_TYPE *a_priority*, size_t *a_stack_size*, emstream * *p_ser_dev*)

This constructor creates a task which controls the running mode and speed of a DC motor using input from **`task_user`** . The main job of this constructor is to call the constructor of parent class (`frt_task`).

Parameters:

a_name	A character string which will be the name of this task
a_priority	The priority at which this task will initially run (default: 0)
a_stack_size	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)
p_ser_dev	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)

Definition at line 56 of file task_motor.cpp.

Member Function Documentation

void task_motor::run (void)

This method is called once by the RTOS scheduler. It constructs the motor driver to run using fast PWM for two separate instances. Each time around the for (;;) loop, the motor driver is updated with the latest shared variables from **task_user**.

Definition at line 71 of file task_motor.cpp.

References Motor::brake(), Motor::freewheel(), and Motor::set_power().

The documentation for this class was generated from the following files:

task_motor.h
task_motor.cpp

task_position Class Reference

This task controls and reads an encoder.

`#include <task_position.h>`

Public Member Functions

task_position (const char *, unsigned portBASE_TYPE, size_t, emstream *)
void **run** (void)

Protected Attributes

int16_t **pos_1**
int16_t **pos_2**
uint16_t **hinge_limit**
uint16_t **threshold**
uint8_t **base_r_limit**
uint8_t **base_l_limit**
uint8_t **runs**
bool **done_1**
bool **done_2**
uint8_t **tol**
uint16_t **high_left**
uint16_t **high_right**
uint16_t **center**
uint16_t **low_left**
uint16_t **low_right**

Detailed Description

This task controls and reads an encoder.

The encoder readed and controler is run using a driver in files

Definition at line 55 of file task_position.h.

Constructor & Destructor Documentation

task_position::task_position (const char * *a_name*, unsigned portBASE_TYPE *a_priority*, size_t *a_stack_size*, emstream * *p_ser_dev*)

This constructor creates a task which reads input from an encoder and controls the encoder using input from **`task_user`** . The main job of this constructor is to call the constructor of parent class (`frt_task`).

Parameters:

a_name	A character string which will be the name of this task
a_priority	The priority at which this task will initially run (default: 0)
a_stack_size	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)

p_ser_dev	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)

Definition at line 46 of file task_position.cpp.

Member Function Documentation

void task_position::run (void)

This method is called once by the RTOS scheduler. It constructs the encoder to run using external interrupts. Each time around the for (;;) loop, the encoder is updated with the latest shared variables from the motor and/or **task_user**.

Definition at line 62 of file task_position.cpp.

References p_print_ser_queue.

The documentation for this class was generated from the following files:

task_position.h
task_position.cpp

task_scan Class Reference

This task controls and reads an encoder.

`#include <task_scan.h>`

Public Member Functions

task_scan (const char *, unsigned portBASE_TYPE, size_t, emstream *)
void **run** (void)

Protected Attributes

uint16_t **R0_C0**
uint16_t **R0_C1**
uint16_t **R0_C2**
uint16_t **R1_C0**
uint16_t **R1_C1**
uint16_t **R1_C2**
uint16_t **R2_C0**
uint16_t **R2_C1**
uint16_t **R2_C2**
uint16_t **column_0**
uint16_t **column_1**
uint16_t **column_2**
uint16_t **row_0**
uint16_t **row_1**
uint16_t **row_2**
int16_t **pos_1**
int16_t **pos_2**
uint16_t **hinge_limit**
uint8_t **hinge_tol**
uint8_t **base_tol**

Detailed Description

This task controls and reads an encoder.

The encoder readed and controler is run using a driver in files

Definition at line 55 of file task_scan.h.

Constructor & Destructor Documentation

task_scan::task_scan (const char * *a_name*, unsigned portBASE_TYPE *a_priority*, size_t *a_stack_size*, emstream * *p_ser_dev*)

This constructor creates a task which reads input from an encoder and controls the encoder using input from **`task_user`** . The main job of this constructor is to call the constructor of parent class (`frt_task`).

Parameters:

a_name	A character string which will be the name of this task
a_priority	The priority at which this task will initially run (default: 0)
a_stack_size	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)
p_ser_dev	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)

Definition at line 45 of file task_scan.cpp.

Member Function Documentation

void task_scan::run (void)

This method is called once by the RTOS scheduler. It constructs the encoder to run using external interrupts. Each time around the for (;;) loop, the encoder is updated with the latest shared variables from the motor and/or **task_user**.

Definition at line 60 of file task_scan.cpp.

References p_print_ser_queue, and adc::read_oversampled().

The documentation for this class was generated from the following files:

task_scan.h
task_scan.cpp

task_sensor Class Reference

This task reads input from the phototransistor sensor array.

`#include <task_sensor.h>`

Public Member Functions

task_sensor (const char *, unsigned portBASE_TYPE, size_t, emstream *)
void **run** (void)

Protected Attributes

uint16_t **high_left**
uint16_t **high_right**
uint16_t **center**
uint16_t **low_left**
uint16_t **low_right**

Detailed Description

This task reads input from the phototransistor sensor array.

The A/D converters for each phototransistor are read and stored in shared data space for use in other tasks.

Definition at line 56 of file task_sensor.h.

Constructor & Destructor Documentation

task_sensor::task_sensor (const char * *a_name*, unsigned portBASE_TYPE *a_priority*, size_t *a_stack_size*, emstream * *p_ser_dev*)

This constructor creates a task which reads input from a phototransistor array The main job of this constructor is to call the constructor of parent class (`frt_task`).

Parameters:

a_name	A character string which will be the name of this task
a_priority	The priority at which this task will initially run (default: 0)
a_stack_size	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)
p_ser_dev	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)

Definition at line 45 of file task_sensor.cpp.

Member Function Documentation

void task_sensor::run (void)

This method is called once by the RTOS scheduler. The A/D converters associated with each phototransistor are oversampled on each pass. The readings are stored in shared variables for use in other tasks.

Definition at line 60 of file task_sensor.cpp.

The documentation for this class was generated from the following files:

task_sensor.h
task_sensor.cpp

task_trigger Class Reference

This task controls PWM of a servo motor used to pull a trigger.

`#include <task_trigger.h>`

Public Member Functions

task_trigger (const char *, unsigned portBASE_TYPE, size_t, emstream *)
void **run** (void)

Protected Attributes

bool **ready**

Detailed Description

This task controls PWM of a servo motor used to pull a trigger.

The PWM for the servo motor is changed whenever the flag for pulling the trigger is high. This flag is pulled high by **task_control**. When the PWM is changed, the trigger pulls.

Definition at line 55 of file task_trigger.h.

Constructor & Destructor Documentation

task_trigger::task_trigger (const char * *a_name*, unsigned portBASE_TYPE *a_priority*, size_t *a_stack_size*, emstream * *p_ser_dev*)

This constructor creates a task which controls the trigger pull of Jankbot using input from **task_control**. The main job of this constructor is to call the constructor of parent class (`frt_task`); the parent's constructor the work.

Parameters:

a_name	A character string which will be the name of this task
a_priority	The priority at which this task will initially run (default: 0)
a_stack_size	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)
p_ser_dev	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)

Definition at line 47 of file task_trigger.cpp.

Member Function Documentation

void task_trigger::run (void)

This method is called once by the RTOS scheduler. Each time around the for (;;) loop, it checks the trigger flag to set PWM for the servo motor to pull the trigger.

Definition at line 64 of file task_trigger.cpp.

The documentation for this class was generated from the following files:

task_trigger.h
task_trigger.cpp

task_user Class Reference

This task interacts with the user to tell him/her how to interface with the system.

`#include <task_user.h>`

Public Member Functions

task_user (const char *, unsigned portBASE_TYPE, size_t, emstream *)
void **run** (void)

Protected Member Functions

void **print_help_message** (void)
void **print_encoder_message** (void)
void **print_motochoice_message** (void)
void **print_direction_message** (void)
void **show_status** (void)

Detailed Description

This task interacts with the user to tell him/her how to interface with the system.

This task sends information to the shared variables that dictate the motors' mode of operation and velocity (when in a running mode only).

Definition at line 61 of file task_user.h.

Constructor & Destructor Documentation

task_user::task_user (const char * *a_name*, unsigned portBASE_TYPE *a_priority*, size_t *a_stack_size*, emstream * *p_ser_dev*)

This constructor creates a new data acquisition task. Its main job is to call the parent class's constructor which does most of the work.

Parameters:

a_name	A character string which will be the name of this task
a_priority	The priority at which this task will initially run (default: 0)
a_stack_size	The size of this task's stack in bytes (default: configMINIMAL_STACK_SIZE)
p_ser_dev	Pointer to a serial device (port, radio, SD card, etc.) which can be used by this task to communicate (default: NULL)

Definition at line 60 of file task_user.cpp.

Member Function Documentation

void task_user::print_help_message (void) [protected]

This method prints a simple help message.

Definition at line 456 of file task_user.cpp.

References PROGRAM_VERSION.

Referenced by run().

void task_user::run (void)

This method is called by the RTOS once to run the task loop for ever and ever.

This task allows the user to control the motors using the PUTTY terminal. If the user is unfamiliar with how to operate the motors using the interface, a help message can be generated by hitting either the "?" or "h" key on his/her keyboard.

Definition at line 78 of file task_user.cpp.

References Encoder::clear_count(), p_print_ser_queue, print_help_message(), Encoder::set_count(), and show_status().

void task_user::show_status (void) [protected]

This method displays information about the status of the system, including the following:

- The name and version of the program
- The name, status, priority, and free stack space of each task
- Processor cycles used by each task
- Amount of heap space free and setting of RTOS tick timer

Definition at line 515 of file task_user.cpp.

References PROGRAM_VERSION.

Referenced by run().

The documentation for this class was generated from the following files:

task_user.h
task_user.cpp

File Documentation

adc.cpp File Reference

```
#include <stdlib.h>
#include <avr/io.h>
#include "rs232int.h"
#include "adc.h"
```

Functions

emstream & **operator<<** (emstream &serpt, **adc** &a2d)

This overloaded operator "prints the A/D converter.".

Detailed Description

This file contains a simple A/D converter driver.

Revisions:

01-15-2008 JRR Original (somewhat useful) file
10-11-2012 JRR Less original, more useful file with FreeRTOS mutex added
10-12-2012 JRR There was a bug in the mutex code, and it has been fixed

License: This file is copyright 2015 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **adc.cpp**.

Function Documentation

emstream& operator<< (emstream & *serpt*, adc & *a2d*)

This overloaded operator "prints the A/D converter.".

This prints the A/D control registers ADCSRA and ADMUX as well as the conversion value from an oversampled conversion with oversampling of 5

Parameters:

serpt	Reference to a serial port to which the printout will be printed
a2d	Reference to the A/D driver which is being printed

Returns:

A reference to the same serial device on which we write information. This is used to string together things to write with << operators

Definition at line 179 of file adc.cpp.

adc.h File Reference

```
#include "emstream.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
```

Classes

class **adc**

*This class **will** run the A/D converter on an AVR processor.* Functions

emstream & **operator<<** (emstream &, **adc** &)

This overloaded operator "prints the A/D converter.".

Detailed Description

This file contains a very simple A/D converter driver. The driver is hopefully thread safe in FreeRTOS due to the use of a mutex to prevent its use by multiple tasks at the same time. There is no protection from priority inversion, however, except for the priority elevation in the mutex.

Revisions:

01-15-2008 JRR Original (somewhat useful) file
10-11-2012 JRR Less original, more useful file with FreeRTOS mutex added
10-12-2012 JRR There was a bug in the mutex code, and it has been fixed

License: This file is copyright 2012 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **adc.h**.

Function Documentation

emstream& operator<< (emstream & *serpt*, adc & *a2d*)

This overloaded operator "prints the A/D converter.".

This prints the A/D control registers ADCSRA and ADMUX as well as the conversion value from an oversampled conversion with oversampling of 5

Parameters:

serpt	Reference to a serial port to which the printout will be printed
a2d	Reference to the A/D driver which is being printed

Returns:

A reference to the same serial device on which we write information. This is used to string together things to write with << operators

Definition at line 179 of file adc.cpp.

encoder_driver.cpp File Reference

```
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <math.h>
#include "rs232int.h"
#include "encoder_driver.h"
#include "shares.h"
```

Functions

emstream & **operator<<** (emstream &serpt, **Encoder** &vroom)
This overloaded operator "prints the encoder".

ISR (INT4_vect)
ISR_ALIAS (INT5_vect, INT4_vect)
ISR (INT6_vect)
ISR_ALIAS (INT7_vect, INT6_vect)

Detailed Description

This file contains a driver for an incremental optical encoder.This driver tracks the encoder count and the number of errors in reading during operation. The encoder ISR code is also located within this file.

Revisions:

02-09-2015 CAL, TT, JH encoder driver class implemented. Original file was a LED brightness control created by JRR

License: This file is copyright 2015 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **encoder_driver.cpp**.

Function Documentation

ISR (INT4_vect)

This interrupt service routine runs whenever an external input pin changes either from low to high or from high to low. The ISR stores the encoder count and determines the direction the encoder is running by checking the current state with the previous state. It also checks that the encoder does not skip a count, if so it increments the error counter.

Definition at line 224 of file encoder_driver.cpp.

emstream& operator<< (emstream & *serpt*, Encoder & *vroom*)

This overloaded operator "prints the encoder".

This prints the relevant encoder registers PORTE. It is useful for debugging purposes but is not utilized in this build of the driver.

Parameters:

serpt	Reference to a serial port to which the printout will be printed
vroom	Reference to the motor driver which is being printed

Returns:

A reference to the same serial device on which we write information. This is used to string together things to write with << operators

Definition at line 205 of file encoder_driver.cpp.

encoder_driver.h File Reference

```
#include "emstream.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
#include "taskshare.h"
#include "textqueue.h"
#include "shares.h"
```

Classes

class **Encoder**

This class will read an encoder connected to the AVR processor by any of the external input pins 4 through 7. Functions

emstream & **operator<<** (emstream &, **Encoder** &)

This overloaded operator "prints the encoder".

Detailed Description

This file contains a driver for an incremental optical encoder.This driver tracks the encoder count and the number of errors in reading during operation. The driver is hopefully thread safe in FreeRTOS due to the use of a mutex to prevent its use by multiple tasks at the same time. There is no protection from priority inversion, however, except for the priority elevation in the mutex.

Revisions:

02-09-2015 CAL, TT, JH encoder driver class implemented. Original file was a LED brightness control created by JRR

License: This file is copyright 2012 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **encoder_driver.h**.

Function Documentation

emstream& operator<< (emstream & *serpt*, Encoder & *vroom*)

This overloaded operator "prints the encoder".

This prints the relevant encoder registers PORTE. It is useful for debugging purposes but is not utilized in this build of the driver.

Parameters:

serpt	Reference to a serial port to which the printout will be printed
vroom	Reference to the motor driver which is being printed

Returns:

A reference to the same serial device on which we write information. This is used to string together things to write with << operators

Definition at line 205 of file encoder_driver.cpp.

main.cpp File Reference

```
#include <stdlib.h>
#include <avr/io.h>
#include <avr/wdt.h>
#include <string.h>
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "croutine.h"
#include "rs232int.h"
#include "time_stamp.h"
#include "taskbase.h"
#include "textqueue.h"
#include "taskqueue.h"
#include "taskshare.h"
#include "shares.h"
#include "task_motor.h"
#include "task_user.h"
#include "encoder_driver.h"
#include "task_encoder.h"
#include "task_control.h"
#include "task_sensor.h"
#include "task_trigger.h"
#include "task_position.h"
```

Functions

int **main** (void)

Variables

TextQueue * **p_print_ser_queue**
TaskShare< int16_t > * **p_share_1**
TaskShare< int16_t > * **p_share_2**
TaskShare< uint8_t > * **p_mode**
TaskShare< uint8_t > * **p_state**
TaskShare< uint8_t > * **p_state_old_1**
TaskShare< uint8_t > * **p_state_old_2**
TaskShare< uint32_t > * **p_error_cntr**
TaskShare< uint32_t > * **p_encoder_cntr_1**
TaskShare< uint32_t > * **p_encoder_cntr_2**
TaskShare< uint8_t > * **p_ext_pin_A**
TaskShare< uint8_t > * **p_ext_pin_B**
TaskShare< uint8_t > * **p_ext_pin_C**
TaskShare< uint8_t > * **p_ext_pin_D**
TaskShare< int16_t > * **p_position_1**
TaskShare< int16_t > * **p_position_2**
TaskShare< bool > * **fire_at_will**
TaskShare< uint16_t > * **p_high_left**
TaskShare< uint16_t > * **p_high_right**
TaskShare< uint16_t > * **p_center**
TaskShare< uint16_t > * **p_low_left**
TaskShare< uint16_t > * **p_low_right**

TaskShare< bool > * **p_pos_done_1**
TaskShare< bool > * **p_pos_done_2**

Detailed Description

This file contains the **main()** code for a program which runs the ME405 board for ME405 Final Project. This program uses an A/D converter to convert an analog signal into an LED brightness via pulse width modulation. It also uses a motor driver to independently control separate motors connected to the VNH3P30 motor drivers of the ME 405 board.

Revisions:

09-30-2012 JRR Original file was a one-file demonstration with two tasks
10-05-2012 JRR Split into multiple files, one for each task plus a main one
10-30-2012 JRR A hopefully somewhat stable version with global queue pointers and the new operator used for most memory allocation
11-04-2012 JRR FreeRTOS Swoop demo program changed to a sweet test suite
01-05-2012 JRR Program reconfigured as ME405 Lab 1 starting point
03-28-2014 JRR Pointers to shared variables and queues changed to references
01-04-2015 JRR Names of share & queue classes changed; allocated with new now
01-26-2015 CAL, TT, JH Updated comments and lab name

Definition in file **main.cpp**.

Function Documentation

int main (void)

The main function sets up the RTOS. Some test tasks are created. Then the scheduler is started up; the scheduler runs until power is turned off or there's a reset.

Returns:

This is a real-time microcontroller program which doesn't return. Ever.

Definition at line 152 of file main.cpp.

References p_print_ser_queue.

Variable Documentation

TextQueue* p_print_ser_queue

This is a print queue, descended from `emstream` so that things can be printed into the queue using the "<<" operator and they'll come out the other end as a stream of characters. It's used by tasks that send things to the user interface task to be printed.

Definition at line 75 of file main.cpp.

Referenced by main(), task_control::run(), task_user::run(), task_position::run(), and task_scan::run().

motor_driver.cpp File Reference

```
#include <stdlib.h>
#include <avr/io.h>
#include <math.h>
#include "rs232int.h"
#include "motor_driver.h"
```

Functions

emstream & **operator<<** (emstream &serpt, **Motor** &vroom)

This overloaded operator "prints the motor driver".

Detailed Description

This file contains a DC motor driver for the VNH3SP30 motor driver attached to the ME 405 board. This driver can control separate motors, causing them to either rotate clockwise, counterclockwise, brake, or freely rotate.

Revisions:

01-15-2008 JRR Original (somewhat useful) file
10-11-2012 JRR Less original, more useful file with FreeRTOS mutex added
10-12-2012 JRR There was a bug in the mutex code, and it has been fixed
01-26-2015 CAL, TT, JH motor driver class implemented

License: This file is copyright 2015 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **motor_driver.cpp**.

Function Documentation

emstream& operator<< (emstream & *serpt*, Motor & *vroom*)

This overloaded operator "prints the motor driver".

This prints the motor driver control registers PORTC, PORTD, PORTB, DDRC DDRD, DDRB, OCR1A, and OCR1B. This is useful for debugging purposes.

Parameters:

serpt	Reference to a serial port to which the printout will be printed
vroom	Reference to the motor driver which is being printed

Returns:

A reference to the same serial device on which we write information. This is used to string together things to write with << operators

Definition at line 171 of file motor_driver.cpp.

motor_driver.h File Reference

```
#include "emstream.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
```

Classes

class **Motor**

*This class **will** operate the VNH3SP30 motor driver on an AVR processor.* Functions

emstream & **operator<<** (emstream &, **Motor** &)

This overloaded operator "prints the motor driver".

Detailed Description

This file contains a DC motor driver for the VNH3SP30 motor driver attached to the ME 405 board. This driver can control separate motors, causing them to either rotate clockwise, counterclockwise, brake, or freely rotate. The driver is hopefully thread safe in FreeRTOS due to the use of a mutex to prevent its use by multiple tasks at the same time. There is no protection from priority inversion, however, except for the priority elevation in the mutex.

Revisions:

01-15-2008 JRR Original (somewhat useful) file
10-11-2012 JRR Less original, more useful file with FreeRTOS mutex added
10-12-2012 JRR There was a bug in the mutex code, and it has been fixed
01-26-2015 CAL, TT, JH motor driver class implemented

Definition in file **motor_driver.h**.

Function Documentation

emstream& operator<< (emstream & *serpt*, Motor & *vroom*)

This overloaded operator "prints the motor driver".

This prints the motor driver control registers PORTC, PORTD, PORTB, DDRC DDRD, DDRB, OCR1A, and OCR1B. This is useful for debugging purposes.

Parameters:

serpt	Reference to a serial port to which the printout will be printed
vroom	Reference to the motor driver which is being printed

Returns:

A reference to the same serial device on which we write information. This is used to string together things to write with << operators

Definition at line 171 of file motor_driver.cpp.

shares.h File Reference

Variables

TextQueue * **p_print_ser_queue**
TaskShare< int16_t > * **p_share_1**
TaskShare< int16_t > * **p_share_2**
TaskShare< uint8_t > * **p_mode**
TaskShare< uint8_t > * **p_state**
TaskShare< uint8_t > * **p_state_old_1**
TaskShare< uint8_t > * **p_state_old_2**
TaskShare< uint32_t > * **p_error_cntr**
TaskShare< uint32_t > * **p_encoder_cntr_1**
TaskShare< uint32_t > * **p_encoder_cntr_2**
TaskShare< uint8_t > * **p_ext_pin_A**
TaskShare< uint8_t > * **p_ext_pin_B**
TaskShare< uint8_t > * **p_ext_pin_C**
TaskShare< uint8_t > * **p_ext_pin_D**
TaskShare< int16_t > * **p_position_1**
TaskShare< int16_t > * **p_position_2**
TaskShare< bool > * **fire_at_will**
TaskShare< uint16_t > * **p_high_left**
TaskShare< uint16_t > * **p_high_right**
TaskShare< uint16_t > * **p_center**
TaskShare< uint16_t > * **p_low_left**
TaskShare< uint16_t > * **p_low_right**
TaskShare< bool > * **p_pos_done_1**
TaskShare< bool > * **p_pos_done_2**

Detailed Description

This file contains extern declarations for queues and other inter-task data communication objects used in a ME405/507/FreeRTOS project.

Revisions:

09-30-2012 JRR Original file was a one-file demonstration with two tasks
10-05-2012 JRR Split into multiple files, one for each task plus a main one
10-29-2012 JRR Reorganized with global queue and shared data references
01-04-2014 JRR Re-reorganized, allocating shares with new now
02-09-2015 JH, TT, CL Added shared variables to be used in multiple tasks and ISRs

Definition in file **shares.h**.

Variable Documentation

TextQueue* p_print_ser_queue

This is a print queue, descended from `emstream` so that things can be printed into the queue using the "<<" operator and they'll come out the other end as a stream of characters. It's used by tasks that send things to the user interface task to be printed.

Definition at line 75 of file main.cpp.

Referenced by main(), task_control::run(), task_user::run(), task_position::run(), and task_scan::run().

task_control.cpp File Reference

```
#include "textqueue.h"
#include "task_control.h"
#include "shares.h"
#include <math.h>
```

Macros

#define **brake_1** 0
#define **free_1** 1
#define **power_1** 2
#define **brake_2** 3
#define **free_2** 4
#define **power_2** 5

Detailed Description

This file contains the code for a task class which controls motor position of an electric motor.

Revisions:

02-17-2015 CAL, TT, JH Original file was a LED brightness control created by JRR. This file has been modified to contain motor control

Definition in file **task_control.cpp**.

task_control.h File Reference

```
#include <stdlib.h>
#include <avr/io.h>
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "taskbase.h"
#include "time_stamp.h"
#include "taskqueue.h"
#include "textqueue.h"
#include "taskshare.h"
#include "rs232int.h"
#include "motor_driver.h"
#include "encoder_driver.h"
#include "emstream.h"
```

Classes

class **task_control**

This task controls and reads a motor with an encoder. Functions

emstream & **operator<<** (emstream &, **task_control** &)

Detailed Description

This file contains the header for a task class that reads and controls an encoder of an electric motor.

Revisions:

02-17-2015 CAL, TT, JH Original file was a LED brightness control created by JRR. This file has been modified to contain motor control

Definition in file **task_control.h**.

task_encoder.cpp File Reference

```
#include "textqueue.h"
#include "task_encoder.h"
#include "shares.h"
```

Detailed Description

This file contains the code for a task class which controls an encoder on an electric motor. It tests the encoder by calling methods from **encoder_driver.cpp**.

Revisions:

02-09-2015 CAL, TT, JH Original file was a LED brightness control created by JRR. This file has been modified to con

License: This file is copyright 2012 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **task_encoder.cpp**.

task_encoder.h File Reference

```
#include <stdlib.h>
#include <avr/io.h>
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "taskbase.h"
#include "time_stamp.h"
#include "taskqueue.h"
#include "taskshare.h"
#include "rs232int.h"
#include "motor_driver.h"
#include "encoder_driver.h"
#include "emstream.h"
```

Classes

class **task_encoder**

This task controls and reads an encoder. Functions

emstream & **operator<<** (emstream &, **Encoder** &)

This overloaded operator "prints the encoder".

Detailed Description

This file contains the header for a task class that reads and controls an encoder on an electric motor.

Revisions:

09-30-2012 JRR Original file was a one-file demonstration with two tasks
10-05-2012 JRR Split into multiple files, one for each task
10-25-2012 JRR Changed to a more fully C++ version with class task_sender
10-27-2012 JRR Altered from data sending task into LED blinking class
11-04-2012 JRR Altered again into the multi-task monstrosity
12-13-2012 JRR Yet again transmogrified; now it controls LED brightness
01-26-2015 CAL, TT, JH encoder driver written using original shell

Definition in file **task_encoder.h**.

Function Documentation

emstream& operator<< (emstream & *serpt*, Encoder & *vroom*)

This overloaded operator "prints the encoder".

This prints the relevant encoder registers PORTE. It is useful for debugging purposes but is not utilized in this build of the driver.

Parameters:

serpt	Reference to a serial port to which the printout will be printed
vroom	Reference to the motor driver which is being printed

Returns:

A reference to the same serial device on which we write information. This is used to string together things to write with << operators

Definition at line 205 of file encoder_driver.cpp.

task_motor.cpp File Reference

```
#include "textqueue.h"
#include "task_motor.h"
#include "shares.h"
```

Macros

#define **brake_1** 0
#define **free_1** 1
#define **power_1** 2
#define **brake_2** 3
#define **free_2** 4
#define **power_2** 5

Detailed Description

This file contains the code for a task class which controls the operation of a DC motor using PWM inputs for speed set by the control loop. The motor's mode of operation is also set by the control loop.

Revisions:

09-30-2012 JRR Original file was a one-file demonstration with two tasks
10-05-2012 JRR Split into multiple files, one for each task
10-25-2012 JRR Changed to a more fully C++ version with class task_sender
10-27-2012 JRR Altered from data sending task into LED blinking class
11-04-2012 JRR Altered again into the multi-task monstrosity
12-13-2012 JRR Yet again transmogrified; now it controls LED brightness
01-26-2015 CAL, TT, JH Original file was a LED brightness control created by JRR

Definition in file **task_motor.cpp**.

task_motor.h File Reference

```
#include <stdlib.h>
#include <avr/io.h>
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "taskbase.h"
#include "time_stamp.h"
#include "taskqueue.h"
#include "taskshare.h"
#include "rs232int.h"
#include "motor_driver.h"
#include "encoder_driver.h"
#include "emstream.h"
```

Classes

class **task_motor**

*This task controls two separate motors using a motor driver and shared variables set in **task_user**.* Functions

emstream & **operator<<** (emstream &, **Motor** &)

This overloaded operator "prints the motor driver".

Detailed Description

This file contains the header for a task class that controls the operation of a DC motor using PWM inputs for speed set by the control loop. The motor's mode of operation is also set by the control loop.

Revisions:

09-30-2012 JRR Original file was a one-file demonstration with two tasks
10-05-2012 JRR Split into multiple files, one for each task
10-25-2012 JRR Changed to a more fully C++ version with class task_sender
10-27-2012 JRR Altered from data sending task into LED blinking class
11-04-2012 JRR Altered again into the multi-task monstrosity
12-13-2012 JRR Yet again transmogrified; now it controls LED brightness
01-26-2015 CAL, TT, JH motor driver class calls

Definition in file **task_motor.h**.

Function Documentation

emstream& operator<< (emstream & *serpt*, Motor & *vroom*)

This overloaded operator "prints the motor driver".

This prints the motor driver control registers PORTC, PORTD, PORTB, DDRC DDRD, DDRB, OCR1A, and OCR1B. This is useful for debugging purposes.

Parameters:

serpt	Reference to a serial port to which the printout will be printed
vroom	Reference to the motor driver which is being printed

Returns:

A reference to the same serial device on which we write information. This is used to string together things to write with << operators

Definition at line 171 of file motor_driver.cpp.

task_position.cpp File Reference

```
#include "task_position.h"
#include "shares.h"
#include <math.h>
```

Detailed Description

This file contains the header for a task class that uses an array of phototransistors to scan for IR light and determine the desired position for the turret based on that data.

Revisions:

03-07-2015 CAL, TT, JH Original file was a LED brightness control created by JRR. This file has been modified to contain position determination and reading of an optical sensor

License: This file is copyright 2012 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **task_position.cpp**.

task_position.h File Reference

```
#include <stdlib.h>
#include <avr/io.h>
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "taskbase.h"
#include "time_stamp.h"
#include "taskqueue.h"
#include "textqueue.h"
#include "taskshare.h"
#include "rs232int.h"
#include "emstream.h"
#include "adc.h"
```

Classes

class **task_position**

This task controls and reads an encoder. Functions

emstream & **operator<<** (emstream &, **task_position** &)

Detailed Description

This file contains the header for a task class that uses an array of phototransistors to scan for IR light.

Revisions:

02-17-2015 CAL, TT, JH Original file was a LED brightness control created by JRR. This file has been modified to contain a scanning system for IR light.

License: This file is copyright 2012 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **task_position.h**.

task_scan.cpp File Reference

```
#include "textqueue.h"
#include "task_scan.h"
#include "shares.h"
#include <math.h>
```

Detailed Description

This file contains the header for a task class that uses an array of phototransistors to scan for IR light.

Revisions:

03-07-2015 CAL, TT, JH Original file was a LED brightness control created by JRR. This file has been modified to contain position determination and reading of an optical sensor

License: This file is copyright 2012 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **task_scan.cpp**.

task_scan.h File Reference

```
#include <stdlib.h>
#include <avr/io.h>
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "taskbase.h"
#include "time_stamp.h"
#include "taskqueue.h"
#include "textqueue.h"
#include "taskshare.h"
#include "rs232int.h"
#include "emstream.h"
#include "adc.h"
```

Classes

class **task_scan**

This task controls and reads an encoder. Functions

emstream & **operator<<** (emstream &, **task_scan** &)

Detailed Description

This file contains the header for a task class that uses an array of phototransistors to scan for IR light.

Revisions:

02-17-2015 CAL, TT, JH Original file was a LED brightness control created by JRR. This file has been modified to contain a scanning system for IR light.

License: This file is copyright 2012 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **task_scan.h**.

task_sensor.cpp File Reference

```
#include "textqueue.h"
#include "task_sensor.h"
#include "shares.h"
#include <math.h>
```

Detailed Description

This file contains the header for a task class that uses an array of phototransistors to scan for IR light.

Revisions:

03-07-2015 CAL, TT, JH Original file was a LED brightness control created by JRR. This file has been modified to contain position determination and reading of an optical sensor

License: This file is copyright 2012 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **task_sensor.cpp**.

task_sensor.h File Reference

```
#include <stdlib.h>
#include <avr/io.h>
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "taskbase.h"
#include "time_stamp.h"
#include "taskqueue.h"
#include "textqueue.h"
#include "taskshare.h"
#include "rs232int.h"
#include "emstream.h"
#include "adc.h"
```

Classes

class **task_sensor**

This task reads input from the phototransistor sensor array. Functions

emstream & **operator<<** (emstream &, **task_sensor** &)

Detailed Description

This file contains the header for a task class that uses an array of phototransistors to scan for IR light.

Revisions:

02-17-2015 CAL, TT, JH Original file was a LED brightness control created by JRR. This file has been modified to contain a scanning system for IR light.

Definition in file **task_sensor.h**.

task_trigger.cpp File Reference

```
#include "textqueue.h"
#include "task_trigger.h"
#include "shares.h"
```

Detailed Description

This file contains the code for a task class which controls the trigger of Jankbot

Revisions:

09-30-2012 JRR Original file was a one-file demonstration with two tasks
10-05-2012 JRR Split into multiple files, one for each task
10-25-2012 JRR Changed to a more fully C++ version with class task_sender
10-27-2012 JRR Altered from data sending task into LED blinking class
11-04-2012 JRR Altered again into the multi-task monstrosity
12-13-2012 JRR Yet again transmogrified; now it controls LED brightness
03-08-2015 CL, JH, & TT change purpose of code; now it controls trigger pull

Definition in file **task_trigger.cpp**.

task_trigger.h File Reference

```
#include <stdlib.h>
#include <avr/io.h>
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "taskbase.h"
#include "time_stamp.h"
#include "taskqueue.h"
#include "taskshare.h"
#include "rs232int.h"
#include "emstream.h"
```

Classes

class **task_trigger**

This task controls PWM of a servo motor used to pull a trigger.

Detailed Description

This file contains the header for a task class that controls trigger of Jankbot

Revisions:

09-30-2012 JRR Original file was a one-file demonstration with two tasks
10-05-2012 JRR Split into multiple files, one for each task
10-25-2012 JRR Changed to a more fully C++ version with class task_sender
10-27-2012 JRR Altered from data sending task into LED blinking class
11-04-2012 JRR Altered again into the multi-task monstrosity
12-13-2012 JRR Yet again transmogrified; now it controls LED brightness

Definition in file **task_trigger.h**.

task_user.cpp File Reference

```
#include <avr/io.h>
#include <avr/wdt.h>
#include "task_user.h"
#include "math.h"
```

Macros

#define **brake_1** 0
#define **free_1** 1
#define **power_1** 2
#define **brake_2** 3
#define **free_2** 4
#define **power_2** 5

Variables

const TickType_t **ticks_to_delay** = ((configTICK_RATE_HZ / 1000) * 5)

Detailed Description

This file contains source code for a user interface task for a ME405/FreeRTOS test suite.

Revisions:

09-30-2012 JRR Original file was a one-file demonstration with two tasks
10-05-2012 JRR Split into multiple files, one for each task
10-25-2012 JRR Changed to a more fully C++ version with class **task_user**
11-04-2012 JRR Modified from the data acquisition example to the test suite
01-04-2014 JRR Changed base class names to TaskBase, TaskShare, etc.
02-09-2015 JH, CL, TT Added user input commands for encoder control. License: This file is copyright 2012 by JR Ridgely and released under the Lesser GNU Public License, version 2. It intended for educational use only, but its use is not limited thereto.

Definition in file **task_user.cpp**.

Variable Documentation

const TickType_t ticks_to_delay = ((configTICK_RATE_HZ / 1000) * 5)

This constant sets how many RTOS ticks the task delays if the user's not talking. The duration is calculated to be about 5 ms.

Definition at line 46 of file task_user.cpp.

task_user.h File Reference

```
#include <stdlib.h>
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "rs232int.h"
#include "time_stamp.h"
#include "taskbase.h"
#include "taskqueue.h"
#include "textqueue.h"
#include "taskshare.h"
#include "shares.h"
#include "encoder_driver.h"
```

Classes

class **task_user**

This task interacts with the user to tell him/her how to interface with the system. Macros

#define **PROGRAM_VERSION** PMS ("ME405 Lab 1 Unmodified Program V0.01 ")

This macro defines a string that identifies the name and version of this program.

Detailed Description

This file contains header stuff for a user interface task for a ME405/FreeRTOS test suite.

Revisions:

09-30-2012 JRR Original file was a one-file demonstration with two tasks
10-05-2012 JRR Split into multiple files, one for each task
10-25-2012 JRR Changed to a more fully C++ version with class **task_user**
11-04-2012 JRR Modified from the data acquisition example to the test suite
01-04-2014 JRR Changed base class names to TaskBase, TaskShare, etc.

Definition in file **task_user.h**

```
//****************************************************************************
/** @file task_user.h
 *    This file contains header stuff for a user interface task for a ME405/FreeRTOS
 *    test suite.
 *
 *  Revisions:
 *    @li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    @li 10-05-2012 JRR Split into multiple files, one for each task
 *    @li 10-25-2012 JRR Changed to a more fully C++ version with class task_user
 *    @li 11-04-2012 JRR Modified from the data acquisition example to the test suite
 *    @li 01-04-2014 JRR Changed base class names to TaskBase, TaskShare, etc.
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//****************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _TASK_USER_H_
#define _TASK_USER_H_

#include <stdlib.h>                         // Prototype declarations for I/O functions

#include "FreeRTOS.h"                       // Primary header for FreeRTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // FreeRTOS inter-task communication queues

#include "rs232int.h"                       // ME405/507 library for serial comm.
#include "time_stamp.h"                     // Class to implement a microsecond timer
#include "taskbase.h"                       // Header for ME405/507 base task class
#include "taskqueue.h"                      // Header of wrapper for FreeRTOS queues
#include "textqueue.h"                      // Header for a "<<" queue class
#include "taskshare.h"                      // Header for thread-safe shared data

#include "shares.h"                         // Global ('extern') queue declarations
#include "encoder_driver.h"                 // Header for encoder_driver file


/// This macro defines a string that identifies the name and version of this program.
#define PROGRAM_VERSION     PMS ("ME405 Lab 1 Unmodified Program V0.01 ")


//---------------------------------------------------------------------------
/** @brief   This task interacts with the user to tell him/her how to interface
 *           with the system.
 *  @details This task sends information to the shared variables that dictate the motors'
 *           mode of operation and velocity (when in a running mode only).
 */

class task_user : public TaskBase
{
    private:
        // No private variables or methods for this class
```

```
    protected:
        // This method displays a simple help message telling the user what to do. It's
        // protected so that only methods of this class or possibly descendents can use it
        void print_help_message (void);

        // This method displays a simple help massage telling the user how to select an
        // encoder operating mode. It's protected so that only methods of this class or
        // possible descendents can use it
        void print_encoder_message (void);

        // This method displays a simple help message telling the user how to select a motor
        // operating mode. It's protected so that only methods of this class or possibly
        // descendents can use it
        void print_motochoice_message (void);

        // This method displays a simple help message telling the user how to select a motor
        // rotation direction. It's protected so that only methods of this class or possibly
        // descendents can use it
        void print_direction_message(void);

        // This method displays information about the status of the system
        void show_status (void);

    public:
        // This constructor creates a user interface task object
        task_user (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        /** This method is called by the RTOS once to run the task loop for ever and ever.
        */
        void run (void);
};

#endif // _TASK_USER_H_
```

```
//****************************************************************************************
/** @file task_user.cpp
 *    This file contains source code for a user interface task for a ME405/FreeRTOS
 *    test suite.
 *
 *  Revisions:
 *    @li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    @li 10-05-2012 JRR Split into multiple files, one for each task
 *    @li 10-25-2012 JRR Changed to a more fully C++ version with class task_user
 *    @li 11-04-2012 JRR Modified from the data acquisition example to the test suite
 *    @li 01-04-2014 JRR Changed base class names to TaskBase, TaskShare, etc.
 *    @li 02-09-2015 JH, CL, TT Added user input commands for encoder control.
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//****************************************************************************************

#include <avr/io.h>                         // Port I/O for SFR's
#include <avr/wdt.h>                        // Watchdog timer header

#include "task_user.h"                      // Header for this file
#include "math.h"                           // Mathmatical operators library

#define brake_1 0                           // These defines help make the code more
#define free_1  1                           // readable. Enumeration data types were
#define power_1 2                           // not implemented for the shared data item
#define brake_2 3                           // p_mode, thus defining the integers in this
#define free_2  4                           // way makes the code easier to read and
#define power_2 5                           // understand.


/** This constant sets how many RTOS ticks the task delays if the user's not talking.
 *  The duration is calculated to be about 5 ms.
 */
const TickType_t ticks_to_delay = ((configTICK_RATE_HZ / 1000) * 5);


//---------------------------------------------------------------------------------------
/** This constructor creates a new data acquisition task. Its main job is to call the
 *  parent class's constructor which does most of the work.
 *  @param a_name A character string which will be the name of this task
 *  @param a_priority The priority at which this task will initially run (default: 0)
 *  @param a_stack_size The size of this task's stack in bytes
 *                      (default: configMINIMAL_STACK_SIZE)
 *  @param p_ser_dev Pointer to a serial device (port, radio, SD card, etc.) which can
 *                   be used by this task to communicate (default: NULL)
 */

task_user::task_user (const char* a_name,
                      unsigned portBASE_TYPE a_priority,
                      size_t a_stack_size,
                      emstream* p_ser_dev
                     )
    : TaskBase (a_name, a_priority, a_stack_size, p_ser_dev)
```

```
{
    // Nothing is done in the body of this constructor. All the work is done in the
    // call to the frt_task constructor on the line just above this one
}


//-------------------------------------------------------------------------------------
/** This task allows the user to control the motors using the PUTTY terminal. If the user
 *  is unfamiliar with how to operate the motors using the interface, a help message can
 *  be generated by hitting either the "?" or "h" key on his/her keyboard.
 */

void task_user::run (void)
{
    char char_in;                           // Character read from serial device
    time_stamp a_time;                      // Holds the time so it can be displayed
    int16_t number_entered = 0;            // Holds a number being entered by user
    uint32_t big_number_entered = 0;        // Holds a number being entered by user
    uint8_t motor = 0;                      // Signifies which motor is being used

    // Tell the user how to get into command mode (state 1), where the user interface
    // task does interesting things such as diagnostic printouts
    *p_serial << PMS ("Press 'h' or '?' for help") << endl;

    // Constructer call for the encoder driver
    Encoder* p_encoder_2 = new Encoder (p_serial, &PORTE, &DDRE, &PINE, ISC60, ISC61, 6, ISC70,
                                    ISC71, 7);

    // This is an infinite loop; it runs until the power is turned off. There is one
    // such loop inside the code for each task
    for (;;)
    {
        // Run the finite state machine. The variable 'state' is kept by parent class
        switch (state)
        {
            // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            // In state 0, we're in command mode, so when the user types characters,
            // the characters are interpreted as commands to do something
            case (0):
                if (p_serial->check_for_char ())            // If the user typed a
                {                                           // character, read
                    char_in = p_serial->getchar ();         // the character

                    // In this switch statement, we respond to different characters as
                    // commands typed in by the user
                    switch (char_in)
                    {
                        // The 'e' command
                        case ('e'):
                            print_encoder_message ();
                            transition_to(5);
                            break;

                        // The '1' command asks to control motor 1
                        case ('1'):
                            print_motochoice_message ();
                            transition_to (2);
                            break;
                        // The '2' command asks to control motor 1
                        case ('2'):
                            print_motochoice_message ();
                            transition_to (3);
                            break;
                        // The 't' command asks what time it is right now
                        case ('t'):
```

```
                    *p_serial << (a_time.set_to_now ()) << endl;
                    break;

                // The 's' command asks for version and status information
                case ('s'):
                    show_status ();
                    break;

                // The 'd' command has all the tasks dump their stacks
                case ('d'):
                    print_task_stacks (p_serial);
                    break;

                // The 'h' command is a plea for help; '?' works also
                case ('h'):
                case ('?'):
                    print_help_message ();
                    break;

                // A control-C character causes the CPU to restart
                case (3):
                    *p_serial << PMS ("Resetting AVR") << endl;
                    wdt_enable (WDTO_120MS);
                    for (;;);
                    break;

                // If character isn't recognized, ask What's That Function?
                default:
                    *p_serial << '"' << char_in << PMS ("\": WTF?") << endl;
                    break;
            } // End switch for characters
        } // End if a character was received

        break; // End of state 0

    // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    // In state 1, wait for user to enter digits and build 'em into a number
    case (1):
        if (p_serial->check_for_char ())        // If the user typed a
        {                                       // character, read
            char_in = p_serial->getchar ();     // the character

            // Respond to numeric characters, Enter or Esc only. Numbers are
            // put into the numeric value we're building up
            if (char_in >= '0' && char_in <= '9')
            {
                *p_serial << char_in;
                number_entered *= 10;
                number_entered += char_in - '0';
            }
            // Carriage return is ignored; the newline character ends the entry
            else if (char_in == 10)
            {
                *p_serial << "\r";
            }
            // Carriage return or Escape ends numeric entry
            else if (char_in == 13 || char_in == 27)
            {
                if(number_entered > 255 || number_entered < 0)
                {
                    number_entered = 255;
                }

                *p_serial << endl << PMS ("Magnitude Set: ")
                          << number_entered << endl;
```

```
                print_direction_message ();
                transition_to (4);
            }
            else
            {}
        }

        // Check the print queue to see if another task has sent this task
        // something to be printed
        else if (p_print_ser_queue->check_for_char ())
        {
            p_serial->putchar (p_print_ser_queue->getchar ());
        }
        break; // End of state 1

    // In state 2, we respond to user input regarding mode selection for motor 1
    case (2):
        if (p_serial->check_for_char ())            // If the user typed a
        {                                           // character, read
            char_in = p_serial->getchar ();         // the character

            // In this switch statement, we respond to different characters as
            // commands typed in by the user
            switch (char_in)
            {
                // The 'b' command asks to brake motor 1
                case('b'):
                    *p_serial << PMS ("Motor 1 is braking (press ? or h for help menu)") << endl;
                    number_entered = 0;
                    transition_to (0);
                    p_mode -> put (brake_1);
                    break;

                // The 'f' command asks to freewheel motor 1
                case('f'):
                    *p_serial << PMS ("Motor 1 is Freewheeling (press ? or h for help menu)") <<
endl;
                    transition_to (0);
                    p_mode->put (free_1);
                    break;

                // The 'r' command asks to run motor 1
                case('r'):
                    *p_serial << PMS ("Set warp drive power for Motor 1") << endl;
                    number_entered = 0;
                    transition_to (1);
                    motor = 1;
                    break;

                // The 'Ctrl-B' command asks to return to previous menu
                case(2):
                    *p_serial << PMS ("Aww... okay... :( (press h or ? for help menu)") << endl;
                    transition_to (0);
                    break;
            }
        }
        break; // end of state 2

    // In state 3, we respond to user input regarding mode selection for motor 2
    case (3):
        if (p_serial->check_for_char ())            // If the user typed a
        {                                           // character, read
            char_in = p_serial->getchar ();         // the character

            // In this switch statement, we respond to different characters as
```

```
            // commands typed in by the user
            switch (char_in)
            {
                // The 'b' command asks to brake motor 2
                case('b'):
                    *p_serial << PMS ("Motor 2 is braking (press ? or h for help menu)") << endl;
                    number_entered = 0;
                    transition_to (0);
                    p_mode -> put (brake_2);
                    break;

                // The 'f' command asks to freewheel motor 2
                case('f'):
                    *p_serial << PMS ("Motor 2 is Freewheeling (press ? or h for help menu)") <<
                endl;
                    transition_to (0);
                    p_mode->put (free_2);
                    break;

                // The 'r' command asks to run motor 2
                case('r'):
                    *p_serial << PMS ("Set warp drive power for Motor 2") << endl;
                    number_entered = 0;
                    transition_to (1);
                    motor = 2;
                    break;

                // The 'Ctrl-B' command asks to return to previous menu
                case(2):
                    *p_serial << PMS ("Aww... okay... :( (press h or ? for help menu)") << endl;
                    transition_to (0);
                    break;
            }
        }
        break; // end of state 3

    // In state 4, we respond to user input regarding desired motor direction
    case(4):
        if (p_serial->check_for_char ())            // If the user typed a
        {                                           // character, read
            char_in = p_serial->getchar ();         // the character

            // In this switch statement, we respond to different characters as
            // commands typed in by the user
            switch (char_in)
            {
                // The 'p' command asks to set motor direction to clockwise
                case('p'):
                    *p_serial << PMS ("All the clockwise! (press h or ? for help menu)") << endl;
                    transition_to (0);

                    if(motor == 1)
                    {
                        p_share_1->put (number_entered);
                        p_mode->put(power_1);
                    }
                    else
                    {
                        p_share_2->put (number_entered);
                        p_mode->put(power_2);
                    }
                    break;

                // The 'n' command asks to set motor direction to counterclockwise
                case('n'):
```

```
                    *p_serial << PMS ("Counter da Clock (press h or ? for help menu)") << endl;
                    transition_to (0);
                    number_entered = 0 - number_entered;

                    if(motor == 1)
                    {
                        p_share_1->put (number_entered);
                        p_mode->put(power_1);
                    }
                    else
                    {
                        p_share_2->put (number_entered);
                        p_mode->put(power_2);
                    }
                    break;

                // The 'Ctrl-B' command asks to return to previous menu
                case(2):
                    *p_serial << PMS ("Set Motor Magnitude:") << endl;
                    number_entered = 0;
                    transition_to (1);
                    break;
            }
        }
        break; // end of state 4

    // In state 5, we respond to user input regarding the encoder
    case (5):
        if (p_serial->check_for_char ())            // If the user typed a
        {                                           // character, read
            char_in = p_serial->getchar ();         // the character

            // In this switch statement, we respond to different characters as
            // commands typed in by the user
            switch (char_in)
            {
                // The 'z' command zeroes the encoder
                case ('z'):
                    *p_serial << PMS ("Encoder is zeroed (press ? or h for help menu)") << endl;
                    p_encoder_2->clear_count(2);
                    transition_to (0);
                    break;

                // The 'v' command sets the encoder count
                case ('v'):
                    big_number_entered = 0;
                    transition_to (6);
                    break;

                // The 'Ctrl-B' command asks to return to previous menu
                case(2):
                    *p_serial << PMS ("Aww... okay... :( (press h or ? for help menu)") << endl;
                    transition_to (0);
                    break;
            }
        }
        break; // end of state 5

                // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    // In state 6, wait for user to enter digits and build 'em into a number for encoder position
    case (6):
        if (p_serial->check_for_char ())        // If the user typed a
        {                                       // character, read
            char_in = p_serial->getchar ();     // the character
```

```
                    // Respond to numeric characters, Enter or Esc only. Numbers are
                    // put into the numeric value we're building up
                    if (char_in >= '0' && char_in <= '9')
                    {
                        *p_serial << char_in;
                        big_number_entered *= 10;
                        big_number_entered += char_in - '0';
                    }
                    // Carriage return is ignored; the newline character ends the entry
                    else if (char_in == 10)
                    {
                        *p_serial << "\r";
                    }
                    // Carriage return or Escape ends numeric entry
                    else if (char_in == 13 || char_in == 27)
                    {
                        if(big_number_entered > 4294967294 || big_number_entered < 0)
                        {
                            big_number_entered = 4294967294;
                        }

                        *p_serial << endl << PMS ("Encoder Set: ")
                                  << big_number_entered << endl;
                        p_encoder_2->set_count(2,big_number_entered);
                        transition_to (0);
                    }
                    else
                    {}
                }

                // Check the print queue to see if another task has sent this task
                // something to be printed
                else if (p_print_ser_queue->check_for_char ())
                {
                    p_serial->putchar (p_print_ser_queue->getchar ());
                }

                break; // End of state 6
            // - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
            // We should never get to the default state. If we do, complain and restart
            default:
                *p_serial << PMS ("Illegal state! Resetting AVR") << endl;
                wdt_enable (WDTO_120MS);
                for (;;);
                break;

        } // End switch state

        // Check the queue to see if another task has sent this task something to print
        if (p_print_ser_queue->check_for_char())
        {
            p_serial->putchar (p_print_ser_queue->getchar());
        }

        runs++;                             // Increment counter for debugging

        // No matter the state, wait for approximately a millisecond before we
        // run the loop again. This gives lower priority tasks a chance to run
        delay_ms (1);
    }
}


//-------------------------------------------------------------------------------
/** This method prints a simple help message.
```

```
 */

void task_user::print_help_message (void)
{
    *p_serial << PROGRAM_VERSION << PMS (" help") << endl;
    *p_serial << PMS ("  1:     Control Motor 1") << endl;
    *p_serial << PMS ("  2:     Control Motor 2") << endl;
    *p_serial << PMS ("  e:     Control Encoder") << endl;
    *p_serial << PMS ("  t:     Show the time right now") << endl;
    *p_serial << PMS ("  s:     Version and setup information") << endl;
    *p_serial << PMS ("  d:     Stack dump for tasks") << endl;
    *p_serial << PMS ("  Ctl-C: Reset the AVR") << endl;
    *p_serial << PMS ("  h:     HALP!") << endl;

}

//-------------------------------------------------------------------------------------
// This method contains the choices for motor control by the user

void task_user::print_encoder_message (void)
{
    *p_serial << endl;
    *p_serial << PMS (" Encoder States") << endl;
    *p_serial << PMS ("  z:     Zero") << endl;
    *p_serial << PMS ("  v:     Set Encoder Count") << endl;
    *p_serial << PMS ("  Ctl-B: Just kidding! GO BACK!") << endl;
}

//-------------------------------------------------------------------------------------
// This method contains the choices for motor control by the user

void task_user::print_motochoice_message (void)
{
    *p_serial << endl;
    *p_serial << PMS (" Motor States") << endl;
    *p_serial << PMS ("  b:     Brake") << endl;
    *p_serial << PMS ("  f:     Freewheel") << endl;
    *p_serial << PMS ("  r:     Run!") << endl;
    *p_serial << PMS ("  Ctl-B: Just kidding! GO BACK!") << endl;
}

//-------------------------------------------------------------------------------------
// This method contains the choices for clockwise/counter clockwise motor direction by the user

void task_user::print_direction_message (void)
{
    *p_serial << endl;
    *p_serial << PMS ("Motor Direction") << endl;
    *p_serial << PMS ("  p:     Clockwise Rotation") << endl;
    *p_serial << PMS ("  n:     Counterclockwise Rotation") << endl;
    *p_serial << PMS ("  Ctl-B: Just kidding! GO BACK!") << endl;
}
//-------------------------------------------------------------------------------------
/** This method displays information about the status of the system, including the
 *  following:
 *    \li The name and version of the program
 *    \li The name, status, priority, and free stack space of each task
 *    \li Processor cycles used by each task
 *    \li Amount of heap space free and setting of RTOS tick timer
 */

void task_user::show_status (void)
{
    time_stamp the_time;                    // Holds current time for printing
```

```
    // First print the program version, compile date, etc.
    *p_serial << endl << PROGRAM_VERSION << PMS (__DATE__) << endl
              << PMS ("System time: ") << the_time.set_to_now ()
              << PMS (", Heap: ") << heap_left() << "/" << configTOTAL_HEAP_SIZE
              #ifdef OCR5A
                << PMS (", OCR5A: ") << OCR5A << endl << endl;
              #elif (defined OCR3A)
                << PMS (", OCR3A: ") << OCR3A << endl << endl;
              #else
                << PMS (", OCR1A: ") << OCR1A << endl << endl;
              #endif

    // Have the tasks print their status; then the same for the shared data items
    print_task_list (p_serial);
    *p_serial << endl;
    print_all_shares (p_serial);
}
```

```
//****************************************************************************************
/** @file task_trigger.h
 *    This file contains the header for a task class that controls trigger of Jankbot
 *
 *  Revisions:
 *    @li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    @li 10-05-2012 JRR Split into multiple files, one for each task
 *    @li 10-25-2012 JRR Changed to a more fully C++ version with class task_sender
 *    @li 10-27-2012 JRR Altered from data sending task into LED blinking class
 *    @li 11-04-2012 JRR Altered again into the multi-task monstrosity
 *    @li 12-13-2012 JRR Yet again transmogrified; now it controls LED brightness
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//****************************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _TASK_TRIGGER_H_
#define _TASK_TRIGGER_H_

#include <stdlib.h>                         // Prototype declarations for I/O functions
#include <avr/io.h>                         // Header for special function registers

#include "FreeRTOS.h"                       // Primary header for FreeRTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // FreeRTOS inter-task communication queues

#include "taskbase.h"                       // ME405/507 base task class
#include "time_stamp.h"                     // Class to implement a microsecond timer
#include "taskqueue.h"                      // Header of wrapper for FreeRTOS queues
#include "taskshare.h"                      // Header for thread-safe shared data

#include "rs232int.h"                       // ME405/507 library for serial comm.

#include "emstream.h"                       // Header for serial ports and devices
//----------------------------------------------------------------------------------------
/** @brief   This task controls PWM of a servo motor used to pull a trigger
 *  @details The PWM for the servo motor is changed whenever the flag for pulling the
 *           trigger is high. This flag is pulled high by task_control. When the PWM is
 *           changed, the trigger pulls.
 */

class task_trigger : public TaskBase
{
private:
    // No private variables or methods for this class

protected:
    // This boolean variable indicates whether the trigger should be pulled or not
    bool ready;

public:
    // This constructor creates a generic task of which many copies can be made
```

```
    task_trigger (const char*, unsigned portBASE_TYPE, size_t, emstream*);

    // This method is called by the RTOS once to run the task loop for ever and ever.
    void run (void);
};

#endif // _TASK_TRIGGER_H_
```

```
//****************************************************************************************
/** @file task_trigger.cpp
 *    This file contains the code for a task class which controls the trigger of Jankbot
 *
 *  Revisions:
 *    @li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    @li 10-05-2012 JRR Split into multiple files, one for each task
 *    @li 10-25-2012 JRR Changed to a more fully C++ version with class task_sender
 *    @li 10-27-2012 JRR Altered from data sending task into LED blinking class
 *    @li 11-04-2012 JRR Altered again into the multi-task monstrosity
 *    @li 12-13-2012 JRR Yet again transmogrified; now it controls LED brightness
 *    @li 03-08-2015 CL, JH, & TT change purpose of code; now it controls trigger pull
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//****************************************************************************************

#include "textqueue.h"                      // Header for text queue class
#include "task_trigger.h"                   // Header for this task
#include "shares.h"                         // Shared inter-task communications


//-------------------------------------------------------------------------------------
/** This constructor creates a task which controls the trigger pull of Jankbot using
 *  input from task_control. The main job of this constructor is to call the
 *  constructor of parent class (\c frt_task ); the parent's constructor the work.
 *  @param a_name A character string which will be the name of this task
 *  @param a_priority The priority at which this task will initially run (default: 0)
 *  @param a_stack_size The size of this task's stack in bytes
 *                      (default: configMINIMAL_STACK_SIZE)
 *  @param p_ser_dev Pointer to a serial device (port, radio, SD card, etc.) which can
 *                   be used by this task to communicate (default: NULL)
 */

task_trigger::task_trigger (const char* a_name,
                            unsigned portBASE_TYPE a_priority,
                            size_t a_stack_size,
                            emstream* p_ser_dev
                           )
    : TaskBase (a_name, a_priority, a_stack_size, p_ser_dev)
{
    // Nothing is done in the body of this constructor. All the work is done in the
    // call to the frt_task constructor on the line just above this one
}


//-------------------------------------------------------------------------------------
/** This method is called once by the RTOS scheduler. Each time around the for (;;)
 *  loop, it checks the trigger flag to set PWM for the servo motor to pull the trigger.
 */

void task_trigger::run (void)
{
```

```
    // Make a variable which will hold times to use for precise task scheduling
    TickType_t previousTicks = xTaskGetTickCount ();

    // Configure counter/timer 1 as a PWM for servo motor.
    DDRB = (1 << 7);

    // The PWM is set to fast mode with ICR1 acting as the upper limit for output compare
    TCCR1A |= (1 << WGM11) | (1 << COM1C1) | (1 << COM1C0);

    // The CS11 and CS10 bits set the prescaler for this timer/counter to run the
    // timer at F_CPU / 64
    TCCR1B |= (1 << WGM12) | (1 << WGM13) | (1 << CS11) | (1 << CS10);

    // This upper limit was calculated to achieve a desired output frequency of 30 Hz
    ICR1 = 7499;

    // Start run timer at 0
    runs = 0;

    // This is the task loop for the servo controlled trigger pull. This loop runs until the
    // power is turned off or something equally dramatic occurs.
    for (;;)
    {

        //Pull trigger is fire_at_will is true and keep it there for 100 runs
        if((ready = fire_at_will -> get()) && (runs < 50))
        {
            OCR1C = 1799;
            runs++;
        }

        // Release trigger and clear runs and fire_at_will
        else if (runs < 100)
        {
            OCR1C = 0;
            runs++;
            fire_at_will -> put(false);
        }

        else
        {
            runs = 0;
        }

        // This is a method we use to cause a task to make one run through its task
        // loop every N milliseconds and let other tasks run at other times
        delay_from_for_ms (previousTicks, 50);

    }
}
```

```
//=============================================================================================
/** @file task_sensor.h
 *    This file contains the header for a task class that uses an array of phototransistors
 *    to scan for IR light.
 *
 *  Revisions:
 *    @li 02-17-2015 CAL, TT, JH Original file was a LED brightness control created
 *                   by JRR. This file has been modified to contain a scanning system
 *                   for IR light.
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//=============================================================================================

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _TASK_SENSOR_H_
#define _TASK_SENSOR_H_

//Include relevent files
#include <stdlib.h>                         // Prototype declarations for I/O functions
#include <avr/io.h>                         // Header for special function registers

#include "FreeRTOS.h"                       // Primary header for FreeRTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // FreeRTOS inter-task communication queues

#include "taskbase.h"                       // ME405/507 base task class
#include "time_stamp.h"                     // Class to implement a microsecond timer
#include "taskqueue.h"                      // Header of wrapper for FreeRTOS queues
#include "textqueue.h"                      // Header for a "<<" queue class
#include "taskshare.h"                      // Header for thread-safe shared data

#include "rs232int.h"                       // ME405/507 library for serial comm.

#include "emstream.h"                       // Header for serial ports and devices
#include "adc.h"                            // Header for A/D converter class

//-----------------------------------------------------------------------------------------
/** @brief   This task reads input from the phototransistor sensor array
 *  @details The A/D converters for each phototransistor are read and stored in shared
 *           data space for use in other tasks.
 */

class task_sensor : public TaskBase
{
    private:
        // No private variables or methods for this class

    protected:
        // Each phototransistor in the array is labeled based on physical location
        // in the array.
        uint16_t high_left;
        uint16_t high_right;
```

```
        uint16_t center;
        uint16_t low_left;
        uint16_t low_right;

    public:
        // This constructor creates a generic task of which many copies can be made
        task_sensor (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        // This method is called by the RTOS once to run the task loop for ever and ever.
        void run (void);
};

// This operator prints the task_sensor (see file task_sensor.cpp for details). It's not
// a part of class task_sensor, but it operates on objects of class task_sensor
emstream& operator << (emstream&, task_sensor&);

#endif // _TASK_SENSOR_H_
```

```
//*****************************************************************************
/** @file task_sensor.cpp
 *    This file contains the header for a task class that uses an array of phototransistors
 *    to scan for IR light.
 *
 *  Revisions:
 *    @li 03-07-2015 CAL, TT, JH Original file was a LED brightness control created
 *                    by JRR. This file has been modified to contain position
 *                    determination and reading of an optical sensor
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************
// This define prevents this .h file from being included multiple times in a .cpp file

#include "textqueue.h"                      // Header for text queue class
#include "task_sensor.h"                        // Header for this task
#include "shares.h"                         // Shared inter-task communications
#include <math.h>                           // Includes math library

//-------------------------------------------------------------------------------
/** This constructor creates a task which reads input from a phototransistor array
 *  The main job of this constructor is to call the constructor of parent class
 *  (\c frt_task ).
 *  @param a_name A character string which will be the name of this task
 *  @param a_priority The priority at which this task will initially run (default: 0)
 *  @param a_stack_size The size of this task's stack in bytes
 *                      (default: configMINIMAL_STACK_SIZE)
 *  @param p_ser_dev Pointer to a serial device (port, radio, SD card, etc.) which can
 *                   be used by this task to communicate (default: NULL)
 */

task_sensor::task_sensor (const char* a_name, unsigned portBASE_TYPE a_priority,
                      size_t a_stack_size, emstream* p_ser_dev)
                      : TaskBase (a_name, a_priority, a_stack_size, p_ser_dev)
{

    // Nothing is done in the body of this constructor. All the work is done in the
    // call to the frt_task constructor on the line just above this one
}

//-------------------------------------------------------------------------------
/** This method is called once by the RTOS scheduler. The A/D converters associated with each
 *  phototransistor are oversampled on each pass. The readings are stored in shared variables for use
 *  in other tasks.
 */

void task_sensor::run (void)
{
    // Make a variable which will hold times to use for precise task scheduling
    TickType_t previousTicks = xTaskGetTickCount ();

    // Create an analog to digital converter driver object and a variable in which to
```

```
    // store its output.
    adc* p_adc = new adc (p_serial);

    // This is the task loop for the sensor task. This loop runs until the
    // power is turned off or something equally dramatic occurs
    for (;;)
    {
        // The A/D converter for each sensor is oversampled in these lines
        center = p_adc -> read_oversampled(0,4);
        high_left = p_adc -> read_oversampled(4,4);
        high_right = p_adc -> read_oversampled(3,4);
        low_left = p_adc -> read_oversampled(2,4);
        low_right = p_adc -> read_oversampled(1,4);

        // The A/D readings are stored in shared variables for use in the control loop
        p_high_left-> put(high_left);
        p_high_right-> put(high_right);
        p_center-> put(center);
        p_low_left-> put(low_left);
        p_low_right-> put(low_right);

        // This is a method we use to cause a task to make one run through its task
        // loop every N milliseconds and let other tasks run at other times
        delay_from_for_ms (previousTicks, 100);
    }
}
```

```
//*****************************************************************************
/** @file task_scan.h
 *    This file contains the header for a task class that uses an array of phototransistors
 *    to scan for IR light.
 *
 *  Revisions:
 *    @li 02-17-2015 CAL, TT, JH Original file was a LED brightness control created
 *                   by JRR. This file has been modified to contain a scanning system
 *                   for IR light.
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _TASK_SCAN_H_
#define _TASK_SCAN_H_

//Include relevent files
#include <stdlib.h>                         // Prototype declarations for I/O functions
#include <avr/io.h>                         // Header for special function registers

#include "FreeRTOS.h"                       // Primary header for FreeRTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // FreeRTOS inter-task communication queues

#include "taskbase.h"                       // ME405/507 base task class
#include "time_stamp.h"                     // Class to implement a microsecond timer
#include "taskqueue.h"                      // Header of wrapper for FreeRTOS queues
#include "textqueue.h"                      // Header for a "<<" queue class
#include "taskshare.h"                      // Header for thread-safe shared data

#include "rs232int.h"                       // ME405/507 library for serial comm.

#include "emstream.h"                       // Header for serial ports and devices
#include "adc.h"                            // Header for A/D converter class

//-----------------------------------------------------------------------------
/** @brief   This task controls and reads an encoder
 *  @details The encoder readed and controler is run using a driver in files
 */

class task_scan : public TaskBase
{
    private:
        // No private variables or methods for this class

    protected:
        // Each phototransistor in the 3x3 array labeled as Row_Column
        uint16_t R0_C0;
        uint16_t R0_C1;
        uint16_t R0_C2;
        uint16_t R1_C0;
```

```
        uint16_t R1_C1;
        uint16_t R1_C2;
        uint16_t R2_C0;
        uint16_t R2_C1;
        uint16_t R2_C2;

        // Column sums
        uint16_t column_0;
        uint16_t column_1;
        uint16_t column_2;

        // Row sums
        uint16_t row_0;
        uint16_t row_1;
        uint16_t row_2;

        // Storage for reference position variable
        int16_t pos_1;
        int16_t pos_2;

        // Maximum position for hinge before rack falls out
        uint16_t hinge_limit;

        // The hinge and base tolerance define the +- tolerance for which the
        // controller has for the reference position error
        uint8_t hinge_tol;
        uint8_t base_tol;

    public:
        // This constructor creates a generic task of which many copies can be made
        task_scan (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        // This method is called by the RTOS once to run the task loop for ever and ever.
        void run (void);
};

// This operator prints the A/D converter (see file adc.cpp for details). It's not
// a part of class adc, but it operates on objects of class adc
// emstream& operator << (emstream&, adc&);
emstream& operator << (emstream&, task_scan&);

#endif // _TASK_SCAN_H_
```

```
//*****************************************************************************
/** @file task_scan.cpp
 *    This file contains the header for a task class that uses an array of phototransistors
 *    to scan for IR light.
 *
 *  Revisions:
 *    @li 03-07-2015 CAL, TT, JH Original file was a LED brightness control created
 *                    by JRR. This file has been modified to contain position
 *                    determination and reading of an optical sensor
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************
// This define prevents this .h file from being included multiple times in a .cpp file

#include "textqueue.h"                      // Header for text queue class
#include "task_scan.h"                      // Header for this task
#include "shares.h"                         // Shared inter-task communications
#include <math.h>                           // Includes math library

//-----------------------------------------------------------------------------
/** This constructor creates a task which reads input from an encoder and controls the
 *  encoder using input from @c task_user. The main job of this constructor is to call the
 *  constructor of parent class (\c frt_task ).
 *  @param a_name A character string which will be the name of this task
 *  @param a_priority The priority at which this task will initially run (default: 0)
 *  @param a_stack_size The size of this task's stack in bytes
 *                      (default: configMINIMAL_STACK_SIZE)
 *  @param p_ser_dev Pointer to a serial device (port, radio, SD card, etc.) which can
 *                   be used by this task to communicate (default: NULL)
 */

task_scan::task_scan (const char* a_name, unsigned portBASE_TYPE a_priority,
                      size_t a_stack_size, emstream* p_ser_dev)
                      : TaskBase (a_name, a_priority, a_stack_size, p_ser_dev)
{

    // Nothing is done in the body of this constructor. All the work is done in the
    // call to the frt_task constructor on the line just above this one
}

//-----------------------------------------------------------------------------
/** This method is called once by the RTOS scheduler. It constructs the encoder
 *  to run using external interrupts. Each time around the for (;;) loop, the encoder
 *  is updated with the latest shared variables from the motor and/or task_user.
 */

void task_scan::run (void)
{
    // Make a variable which will hold times to use for precise task scheduling
    TickType_t previousTicks = xTaskGetTickCount ();

    // Initialize PORTA pins 0-7 and PORTE pin 0 as outputs
    DDRE |= (1<<PE0) /*| (1<<PE1) | (1<<PE2)*/;

    // Create an analog to digital converter driver object and a variable in which to
    // store its output.
    adc* p_adc = new adc (p_serial);
```

```
    hinge_limit = 1100;
    hinge_tol = 1000;
    base_tol = 1000;

    // This is the task loop for the encoder control task. This loop runs until the
    // power is turned off or something equally dramatic occurs
    for (;;)
    {
        // Read first column by turning on PORTE pin 0
        PORTE |= (1<<PE0);
        R0_C0 = p_adc->read_oversampled(0,1);
        R1_C0 = p_adc->read_oversampled(0,1);
        R2_C0 = p_adc->read_oversampled(0,1);
        PORTE &= ~(1<<PE0);

        // Read first column by turning on PORTE pin 1
        PORTE |= (1<<PE0);
        R0_C1 = p_adc->read_oversampled(0,1);
        R1_C1 = p_adc->read_oversampled(0,1);
        R2_C1 = p_adc->read_oversampled(0,1);
        PORTE &= ~(1<<PE0);

        // Read first column by turning on PORTE pin 2
        PORTE |= (1<<PE0);
        R0_C2 = p_adc->read_oversampled(0,1);
        R1_C2 = p_adc->read_oversampled(0,1);
        R2_C2 = p_adc->read_oversampled(0,1);
        PORTE &= ~(1<<PE0);

        // sum columns and compare to determine rotation needed
        column_0 = R0_C0 + R1_C0 + R2_C0;
        column_1 = R0_C1 + R1_C1 + R2_C1;
        column_2 = R0_C2 + R1_C2 + R2_C2;

        // sum rows and compare to determine tilt needed
        row_0 = R0_C0 + R0_C1 + R0_C2;
        row_1 = R1_C0 + R1_C1 + R1_C2;
        row_2 = R2_C0 + R2_C1 + R2_C2;

        pos_1 = p_position_1 -> get();
        pos_2 = p_position_2 -> get();

        if (pos_1 < hinge_limit || pos_1 >= 0)
        {
            if (((row_0 >= (row_1 - hinge_tol)) && (row_0 <= (row_1 + hinge_tol))) ||
                ((row_1 >= (row_2 - hinge_tol)) && (row_1 <= (row_2 + hinge_tol))))
            {
                p_position_1 -> put(pos_1);
            }

            else if ((row_2 < row_0) || (row_1 < row_0))
            {
                // tilt down
                pos_1++;
                p_position_1 -> put(pos_1);
            }

            else if ((row_2 > row_0) || (row_1 > row_0))
            {
                // tilt up
                pos_1--;
                p_position_1 -> put(pos_1);
            }
        }

        // Compare Columns to determine rotation needed
        if (((column_0 >= (column_1 - base_tol)) && (column_0 <= (column_1 + base_tol))) ||
            ((column_2 >= (column_1 - base_tol)) && (column_2 <= (column_1 + base_tol))))
        {
```

```
            p_position_2 -> put(pos_2);
        }

        else if ((column_0 > column_2) || (column_0 > column_1))
        {
            // need to spin to the right
            pos_2++;
            p_position_2 -> put(pos_2);
        }

        else if ((column_0 < column_2) || (column_0 < column_1))
        {
            // spin to the left
            pos_2--;
            p_position_2 -> put(pos_2);
        }

        *p_print_ser_queue  << "R0_C0: " << dec << R0_C0 << "   R0_C1: " << R0_C1 << "   R0_C2: " << R0_C2
        << endl;
        *p_print_ser_queue  << "R1_C0: " << R1_C0 << "   R1_C1: " << R1_C1 << "   R1_C2: " << R1_C2 <<
        endl;
        *p_print_ser_queue  << "R2_C0: " << R2_C0 << "   R2_C1: " << R2_C1 << "   R2_C2: " << R2_C2 <<
        endl << endl;

//      *p_print_ser_queue  << "Row 0: " << row_0 << endl;
//      *p_print_ser_queue  << "Row 1: " << row_1 << endl;
//      *p_print_ser_queue  << "Row 2: " << row_2 << endl <<endl;
//
//      *p_print_ser_queue  << "Column_0: " << column_0 << endl;
//      *p_print_ser_queue  << "Column_1: " << column_1 << endl;
//      *p_print_ser_queue  << "Column_2: " << column_2 << endl << endl;
//
//      *p_print_ser_queue  << "Pos 1: " << pos_1 << endl;
//      *p_print_ser_queue  << "Pos 2: " << pos_2 << endl << endl;
//
        // This is a method we use to cause a task to make one run through its task
        // loop every N milliseconds and let other tasks run at other times
        delay_from_for_ms (previousTicks, 100);
    }
}
```

```
//*****************************************************************************
/** @file task_position.h
 *    This file contains the header for a task class that uses an array of phototransistors
 *    to scan for IR light.
 *
 *  Revisions:
 *    @li 02-17-2015 CAL, TT, JH Original file was a LED brightness control created
 *                   by JRR. This file has been modified to contain a scanning system
 *                   for IR light.
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _TASK_POSITION_H_
#define _TASK_POSITION_H_

//Include relevent files
#include <stdlib.h>                         // Prototype declarations for I/O functions
#include <avr/io.h>                         // Header for special function registers

#include "FreeRTOS.h"                       // Primary header for FreeRTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // FreeRTOS inter-task communication queues

#include "taskbase.h"                       // ME405/507 base task class
#include "time_stamp.h"                     // Class to implement a microsecond timer
#include "taskqueue.h"                      // Header of wrapper for FreeRTOS queues
#include "textqueue.h"                      // Header for a "<<" queue class
#include "taskshare.h"                      // Header for thread-safe shared data

#include "rs232int.h"                       // ME405/507 library for serial comm.

#include "emstream.h"                       // Header for serial ports and devices
#include "adc.h"                            // Header for A/D converter class

//-----------------------------------------------------------------------------
/** @brief   This task controls and reads an encoder
 *  @details The encoder readed and controler is run using a driver in files
 */

class task_position : public TaskBase
{
    private:
        // No private variables or methods for this class

    protected:
        // Storage for reference position variable
        int16_t pos_1;
        int16_t pos_2;

        // Maximum position for hinge before rack falls out
```

```
        uint16_t hinge_limit;

        // Threshold for light detection
        uint16_t threshold;

        // Limits for directions
        uint8_t base_r_limit;
        uint8_t base_l_limit;

        // Run count for scanner
        uint8_t runs;

        // Done flags
        bool done_1;
        bool done_2;

        // Center tolerance
        uint8_t tol;

        // Each phototransistor in the array labeled as Row_Column
        uint16_t high_left;
        uint16_t high_right;
        uint16_t center;
        uint16_t low_left;
        uint16_t low_right;
    public:
        // This constructor creates a generic task of which many copies can be made
        task_position (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        // This method is called by the RTOS once to run the task loop for ever and ever.
        void run (void);
};

// This operator prints the A/D converter (see file adc.cpp for details). It's not
// a part of class adc, but it operates on objects of class adc
// emstream& operator << (emstream&, adc&);
emstream& operator << (emstream&, task_position&);

#endif // _TASK_POSITION_H_
```

```
//*****************************************************************************
/** @file task_position.cpp
 *    This file contains the header for a task class that uses an array of phototransistors
 *    to scan for IR light and determine the desired position for the turret based on that
 *    data.
 *
 *  Revisions:
 *    @li 03-07-2015 CAL, TT, JH Original file was a LED brightness control created
 *                    by JRR. This file has been modified to contain position
 *                    determination and reading of an optical sensor
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************
// This define prevents this .h file from being included multiple times in a .cpp file

// #include "textqueue.h"                     // Header for text queue class
#include "task_position.h"                    // Header for this task
#include "shares.h"                          // Shared inter-task communications
#include <math.h>                            // Includes math library

//-----------------------------------------------------------------------------
/** This constructor creates a task which reads input from an encoder and controls the
 *  encoder using input from @c task_user. The main job of this constructor is to call the
 *  constructor of parent class (\c frt_task ).
 *  @param a_name A character string which will be the name of this task
 *  @param a_priority The priority at which this task will initially run (default: 0)
 *  @param a_stack_size The size of this task's stack in bytes
 *                      (default: configMINIMAL_STACK_SIZE)
 *  @param p_ser_dev Pointer to a serial device (port, radio, SD card, etc.) which can
 *                   be used by this task to communicate (default: NULL)
 */

task_position::task_position (const char* a_name, unsigned portBASE_TYPE a_priority,
                      size_t a_stack_size, emstream* p_ser_dev)
                      : TaskBase (a_name, a_priority, a_stack_size, p_ser_dev)
{

    // Nothing is done in the body of this constructor. All the work is done in the
    // call to the frt_task constructor on the line just above this one
    state = 0;
}

//-----------------------------------------------------------------------------
/** This method is called once by the RTOS scheduler. It constructs the encoder
 *  to run using external interrupts. Each time around the for (;;) loop, the encoder
 *  is updated with the latest shared variables from the motor and/or task_user.
 */

void task_position::run (void)
{
    // Make a variable which will hold times to use for precise task scheduling
    TickType_t previousTicks = xTaskGetTickCount ();
```

```
    state = 0;
    hinge_limit = 500;
    threshold = 10;
    base_r_limit = 600;
    base_l_limit = 1000;
    runs = 0;
    tol = 50;
    p_pos_done_1 -> put(false);
    p_pos_done_2 -> put(false);

    // This is the task loop for the position task. This loop runs until the
    // power is turned off or something equally dramatic occurs
    for (;;)
    {
        switch (state)
        {
            // Spin 160 degrees from start position to face the target area
            case (0):
                p_position_1 -> put(0);
                p_position_2 -> put(700);
                done_1 = p_pos_done_1 -> get();
                done_2 = p_pos_done_2 -> get();
                if ((done_1 == true) && (done_2 == true))
                {
                    transition_to (1);
                }
                break;

            // Search for light source in target area
            case (1):
                // Gather shared variables into local variables
                high_left = p_high_left -> get();
                high_right = p_high_right-> get();
                center = p_center-> get();
                low_left = p_low_left-> get();
                low_right = p_low_right-> get();

                pos_1 = p_position_1 -> get();
                pos_2 = p_position_2 -> get();

                done_1 = p_pos_done_1 -> get();
                done_2 = p_pos_done_2 -> get();

                // If any sensor reads above the threshold transition to state 2
                if ((high_left >= threshold) || (high_right >= threshold) ||
                    (center >= threshold) || (low_left >= threshold) ||
                    (low_right >= threshold))
                {
                    transition_to (2);
                }

                if (done_1==true && done_2==true)
                {
                    p_pos_done_1-> put(false);
                    p_pos_done_2-> put(false);

                    // If above hinge limit, go down
                    if (pos_1 >= hinge_limit)
                    {
                        pos_1 -= 10;
                        p_position_1 -> put(pos_1);
                    }

                    // search pattern
```

```
            else
            {
                pos_1 += 50;
                if (runs == 0)
                {
                    pos_2 += 100;
                    if (pos_2 >= base_l_limit)
                    {
                        runs = 1;
                    }
                }
                else
                {
                    pos_2 -= 100;
                    if (pos_2 <= base_r_limit)
                    {
                        runs = 0;
                    }
                }
            }

            p_position_1 -> put(pos_1);
            p_position_2 -> put(pos_2);
        }

        break;

    // Lock on to target once the light source is found
    case (2):
        // Gather shared variables into local variables
        high_left = p_high_left -> get();
        high_right = p_high_right-> get();
        center = p_center-> get();
        low_left = p_low_left-> get();
        low_right = p_low_right-> get();

        pos_1 = p_position_1 -> get();
        pos_2 = p_position_2 -> get();

        done_1 = p_pos_done_1 -> get();
        done_2 = p_pos_done_2 -> get();

        // Wait until task_control sets done flags, indicating the reference
        // position has been reached
        if (done_1==true && done_2==true)
        {
            p_pos_done_1-> put(false);
            p_pos_done_2-> put(false);

            // Pull trigger if in final position
            if ((center > (low_right - tol)) && (center > (high_right - tol))
                && (center > (low_left - tol)) && (center > (low_right - tol))
                && (center < (low_right + tol)) && (center < (high_right + tol))
                && (center < (low_left - tol)) && (center < (low_right + tol)))
            {
                fire_at_will->put(true);
            }

            else if((high_left> center) || (low_left> center) )
            {
                pos_2+=50;
            }

            else if((high_right> center) || (low_right> center) )
            {
```

```
                    pos_2-=50;
                }

                else if((high_right> center) || (high_left> center) )
                {
                    pos_1+=10;
                }

                else if((low_right> center) || (low_left> center) )
                {
                    pos_1-=10;
                }
                else
                {
                    *p_print_ser_queue << "Borked" << endl;
                }
                p_position_1 -> put(pos_1);
                p_position_2 -> put(pos_2);
            }
            break;

        default:
            *p_print_ser_queue  << "Borked" << endl;
            break;
    }

    // This is a method we use to cause a task to make one run through its task
    // loop every N milliseconds and let other tasks run at other times
    delay_from_for_ms (previousTicks, 50);
  }
}
```

```
//****************************************************************************************
/** @file task_motor.h
 *    This file contains the header for a task class that controls the operation of a
 *    DC motor using PWM inputs for speed set by the control loop. The motor's mode of
 *    operation is also set by the control loop.
 *
 *  Revisions:
 *    @li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    @li 10-05-2012 JRR Split into multiple files, one for each task
 *    @li 10-25-2012 JRR Changed to a more fully C++ version with class task_sender
 *    @li 10-27-2012 JRR Altered from data sending task into LED blinking class
 *    @li 11-04-2012 JRR Altered again into the multi-task monstrosity
 *    @li 12-13-2012 JRR Yet again transmogrified; now it controls LED brightness
 *    @li 01-26-2015 CAL, TT, JH motor driver class calls
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//****************************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _TASK_MOTOR_H_
#define _TASK_MOTOR_H_

#include <stdlib.h>                         // Prototype declarations for I/O functions
#include <avr/io.h>                         // Header for special function registers

#include "FreeRTOS.h"                       // Primary header for FreeRTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // FreeRTOS inter-task communication queues

#include "taskbase.h"                       // ME405/507 base task class
#include "time_stamp.h"                     // Class to implement a microsecond timer
#include "taskqueue.h"                      // Header of wrapper for FreeRTOS queues
#include "taskshare.h"                      // Header for thread-safe shared data

#include "rs232int.h"                       // ME405/507 library for serial comm.
#include "motor_driver.h"                   // Header for Motor driver class
#include "encoder_driver.h"                 // Header for Encoder driver class

#include "emstream.h"                       // Header for serial ports and devices

//--------------------------------------------------------------------------------------
/** @brief   This task controls two separate motors using a motor driver and shared
 *           variables set in task_user
 *  @details The motor controller is run using a driver in files @c motor_driver.h and
 *           @c motor_driver.cpp. Code in this task sets up a timer/counter in PWM mode
 *           and controls the motors' modes and run speeds.
 */

class task_motor : public TaskBase
{
    private:
        // No private variables or methods for this class
```

```
    protected:
        uint8_t mode;
        int16_t speed_1;
        int16_t speed_2;

    public:
        // This constructor creates a generic task of which many copies can be made
        task_motor (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        // This method is called by the RTOS once to run the task loop for ever and ever.
        void run (void);
};

// This operator prints the A/D converter (see file adc.cpp for details). It's not
// a part of class adc, but it operates on objects of class adc
// emstream& operator << (emstream&, adc&);
emstream& operator << (emstream&, Motor&);

#endif // _TASK_MOTOR_H_
```

```
//****************************************************************************************
/** @file task_motor.cpp
 *    This file contains the code for a task class which controls the operation of a
 *    DC motor using PWM inputs for speed set by the control loop. The motor's mode of
 *    operation is also set by the control loop.
 *
 *  Revisions:
 *    @li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    @li 10-05-2012 JRR Split into multiple files, one for each task
 *    @li 10-25-2012 JRR Changed to a more fully C++ version with class task_sender
 *    @li 10-27-2012 JRR Altered from data sending task into LED blinking class
 *    @li 11-04-2012 JRR Altered again into the multi-task monstrosity
 *    @li 12-13-2012 JRR Yet again transmogrified; now it controls LED brightness
 *    @li 01-26-2015 CAL, TT, JH Original file was a LED brightness control created
 *                    by JRR
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//****************************************************************************************

#include "textqueue.h"                      // Header for text queue class
#include "task_motor.h"                     // Header for this task
#include "shares.h"                         // Shared inter-task communications

#define brake_1 0                           // These defines help make the code more
#define free_1  1                           // readable. Enumeration data types were
#define power_1 2                           // not implemented for the shared data item
#define brake_2 3                           // p_mode, thus defining the integers in this
#define free_2  4                           // way makes the code easier to read and
#define power_2 5                           // understand.

//-------------------------------------------------------------------------------------
/** This constructor creates a task which controls the running mode and speed of a DC
 *  motor using input from @c task_user. The main job of this constructor is to call the
 *  constructor of parent class (\c frt_task ).
 *  @param a_name A character string which will be the name of this task
 *  @param a_priority The priority at which this task will initially run (default: 0)
 *  @param a_stack_size The size of this task's stack in bytes
 *                      (default: configMINIMAL_STACK_SIZE)
 *  @param p_ser_dev Pointer to a serial device (port, radio, SD card, etc.) which can
 *                   be used by this task to communicate (default: NULL)
 */

task_motor::task_motor (const char* a_name, unsigned portBASE_TYPE a_priority,
                        size_t a_stack_size, emstream* p_ser_dev)
                        : TaskBase (a_name, a_priority, a_stack_size, p_ser_dev)
{
    // Nothing is done in the body of this constructor. All the work is done in the
    // call to the frt_task constructor on the line just above this one
}


//-------------------------------------------------------------------------------------
```

```
/** This method is called once by the RTOS scheduler. It constructs the motor driver
 *  to run using fast PWM for two separate instances. Each time around the for (;;)
 *  loop, the motor driver is updated with the latest shared variables from task_user.
 */

void task_motor::run (void)
{
    // Make a variable which will hold times to use for precise task scheduling
    TickType_t previousTicks = xTaskGetTickCount ();

    // The following lines of code construct the motor driver for two separate instances
    // of the motor_driver. Each instance controls a different motor. p_motor_1 controls
    // the gun angle
    Motor* p_motor_1 = new Motor (p_serial, &PORTC, &DDRC, 0, &PORTC, &DDRC, 1, &PORTC,
                                  &DDRC, 2, &PORTB, &DDRB, 6, &OCR1B);

    // p_motor_2 drives the base rotations
    Motor* p_motor_2 = new Motor (p_serial, &PORTD, &DDRD, 5, &PORTD, &DDRD, 6, &PORTD,
                                  &DDRD, 7, &PORTB, &DDRB, 5, &OCR1A);


    // These lines configure fast 8-bit fast PWM for motor 2
    TCCR1A |= (1 << WGM10)
            | (1 << COM1A1) | (1 << COM1B1);
    TCCR1A &= ~(1 << COM1A0) & ~(1 << COM1B0);

    // The CS11 and CS10 bits set the prescaler for this timer/counter to run the
    // timer at F_CPU / 64
    TCCR1B |= (1 << WGM12)
            | (1 << CS11)  | (1 << CS10);

    // To set 8-bit fast PWM mode we must set bits WGM30 and WGM32, which are in two
    // different registers (ugh). We use COM3B1 and Com3B0 to set up the PWM so that
    // the pin output will have inverted sense, that is, a 0 is on and a 1 is off;
    // this is needed because the LED connects from Vcc to the pin.
    TCCR3A |= (1 << WGM30)
            | (1 << COM3B1) | (1 << COM3B0);

    // The CS31 and CS30 bits set the prescaler for this timer/counter to run the
    // timer at F_CPU / 64
    TCCR3B |= (1 << WGM32)
            | (1 << CS31)  | (1 << CS30);

    // This is the task loop for the motor control task. This loop runs until the
    // power is turned off or something equally dramatic occurs
    for (;;)
    {
        // The shared variables are stored locally for use as inputs to the motor driver
        mode = p_mode->get();
        speed_1 = p_share_1->get();
        speed_2 = p_share_2->get();

        // Motor 1 is controlled using modes 0 through 2, which correspond to brake_1,
        // free_1, and power_1 respectively. Speed is only used as an input to the method
        // set_power.
        if(mode < 3)
        {
            switch(mode)
            {
                case(brake_1):
                    p_motor_1->brake();
                    break;

                case(free_1):
                    p_motor_1->freewheel();
```

```
                    break;

                case(power_1):
                    p_motor_1->set_power(speed_1);
                    break;

                default:
                    DBG (p_serial, "ERROR...ERROR... Abandon hope" << endl);
                    break;
            }
        }
        // Motor 2 is controlled using modes 3 through 5, which correspond to brake_2,
        // free_2, and power_2 respectively. Speed is only used as an input to the method
        // set_power.
        else
        {
            switch(mode)
            {
                case(brake_2):
                    p_motor_2->brake();
                    break;

                case(free_2):
                    p_motor_2->freewheel();
                    break;

                case(power_2):
                    p_motor_2->set_power(speed_2);
                    break;

                default:
                    DBG (p_serial, "ERROR...ERROR... Abandon hope" << endl);
                    break;
            }
        }

        // This enables motor driver to print debug messages
        *p_serial << (*p_serial, *p_motor_1);
        *p_serial << (*p_serial, *p_motor_2);

        // This is a method we use to cause a task to make one run through its task
        // loop every N milliseconds and let other tasks run at other times
        delay_from_for_ms (previousTicks, 100);
    }
}
```

```
//****************************************************************************************
/** @file task_encoder.h
 *    This file contains the header for a task class that reads and controls an encoder
 *    on an electric motor.
 *
 *  Revisions:
 *    @li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    @li 10-05-2012 JRR Split into multiple files, one for each task
 *    @li 10-25-2012 JRR Changed to a more fully C++ version with class task_sender
 *    @li 10-27-2012 JRR Altered from data sending task into LED blinking class
 *    @li 11-04-2012 JRR Altered again into the multi-task monstrosity
 *    @li 12-13-2012 JRR Yet again transmogrified; now it controls LED brightness
 *    @li 01-26-2015 CAL, TT, JH encoder driver written using original shell
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//****************************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _TASK_ENCODER_H_
#define _TASK_ENCODER_H_

//Include relevent files
#include <stdlib.h>                         // Prototype declarations for I/O functions
#include <avr/io.h>                         // Header for special function registers

#include "FreeRTOS.h"                       // Primary header for FreeRTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // FreeRTOS inter-task communication queues

#include "taskbase.h"                       // ME405/507 base task class
#include "time_stamp.h"                     // Class to implement a microsecond timer
#include "taskqueue.h"                      // Header of wrapper for FreeRTOS queues
#include "taskshare.h"                      // Header for thread-safe shared data

#include "rs232int.h"                       // ME405/507 library for serial comm.
#include "motor_driver.h"                   // Header for Motor driver class
#include "encoder_driver.h"                 // Header for Encoder driver class

#include "emstream.h"                       // Header for serial ports and devices

//--------------------------------------------------------------------------------------
/** @brief   This task controls and reads an encoder
 *  @details The encoder readed and controler is run using a driver in files @c encoder_driver.h and
 *           @c encoder_driver.cpp.
 */

class task_encoder : public TaskBase
{
    private:
        // No private variables or methods for this class

    protected:
```

```
        // No protected variables or methods for this class

    public:
        // This constructor creates a generic task of which many copies can be made
        task_encoder (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        // This method is called by the RTOS once to run the task loop for ever and ever.
        void run (void);
};

// This operator prints the A/D converter (see file adc.cpp for details). It's not
// a part of class adc, but it operates on objects of class adc
// emstream& operator << (emstream&, adc&);
emstream& operator << (emstream&, Encoder&);

#endif // _TASK_ENCODER_H_
```

```
//*****************************************************************************
/** @file task_encoder.cpp
 *    This file contains the code for a task class which controls an encoder
 *    on an electric motor. It tests the encoder by calling methods from encoder_driver.cpp.
 *
 *  Revisions:
 *    @li 02-09-2015 CAL, TT, JH Original file was a LED brightness control created
 *                    by JRR. This file has been modified to con
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************

#include "textqueue.h"                      // Header for text queue class
#include "task_encoder.h"                   // Header for this task
#include "shares.h"                         // Shared inter-task communications

//-----------------------------------------------------------------------------
/** This constructor creates a task which reads input from an encoder and controls the
 *  encoder using input from @c task_user. The main job of this constructor is to call the
 *  constructor of parent class (\c frt_task ).
 *  @param a_name A character string which will be the name of this task
 *  @param a_priority The priority at which this task will initially run (default: 0)
 *  @param a_stack_size The size of this task's stack in bytes
 *                      (default: configMINIMAL_STACK_SIZE)
 *  @param p_ser_dev Pointer to a serial device (port, radio, SD card, etc.) which can
 *                   be used by this task to communicate (default: NULL)
 */

task_encoder::task_encoder (const char* a_name, unsigned portBASE_TYPE a_priority,
                            size_t a_stack_size, emstream* p_ser_dev)
                            : TaskBase (a_name, a_priority, a_stack_size, p_ser_dev)
{
    // Nothing is done in the body of this constructor. All the work is done in the
    // call to the frt_task constructor on the line just above this one
}


//-----------------------------------------------------------------------------
/** This method is called once by the RTOS scheduler. It constructs the encoder
 *  to run using external interrupts. Each time around the for (;;) loop, the encoder
 *  is updated with the latest shared variables from the motor, control loop, and/or task_user.
 */

void task_encoder::run (void)
{
    // Make a variable which will hold times to use for precise task scheduling
    TickType_t previousTicks = xTaskGetTickCount ();

    // Constructer call for the encoder driver, p_encoder_1 measures the gun angle
    Encoder* p_encoder_1 = new Encoder (p_serial, &PORTE, &DDRE, &PINE, ISC40, ISC41, 4, ISC50,
                                        ISC51, 5);
```

```
    // p_encoder_2 measures the base rotation
    Encoder* p_encoder_2 = new Encoder (p_serial, &PORTE, &DDRE, &PINE, ISC60, ISC61, 6, ISC70,
                                        ISC71, 7);

    // This is the task loop for the encoder task. This loop runs until the
    // power is turned off or something equally dramatic occurs
    for (;;)
    {
        // This is a method we use to cause a task to make one run through its task
        // loop every N milliseconds and let other tasks run at other times
        delay (100);
    }
}
```

```
//****************************************************************************************
/** @file task_control.h
 *    This file contains the header for a task class that reads and controls an encoder
 *    of an electric motor.
 *
 *  Revisions:
 *    @li 02-17-2015 CAL, TT, JH Original file was a LED brightness control created
 *                    by JRR. This file has been modified to contain motor control
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//****************************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _TASK_CONTROL_H_
#define _TASK_CONTROL_H_

//Include relevent files
#include <stdlib.h>                         // Prototype declarations for I/O functions
#include <avr/io.h>                         // Header for special function registers

#include "FreeRTOS.h"                       // Primary header for FreeRTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // FreeRTOS inter-task communication queues

#include "taskbase.h"                       // ME405/507 base task class
#include "time_stamp.h"                     // Class to implement a microsecond timer
#include "taskqueue.h"                      // Header of wrapper for FreeRTOS queues
#include "textqueue.h"                      // Header for a "<<" queue class
#include "taskshare.h"                      // Header for thread-safe shared data

#include "rs232int.h"                       // ME405/507 library for serial comm.
#include "motor_driver.h"                   // Header for Motor driver class
#include "encoder_driver.h"                 // Header for Encoder driver class

#include "emstream.h"                       // Header for serial ports and devices

//-------------------------------------------------------------------------------------
/** @brief   This task controls and reads a motor with an encoder
 *  @details The encoder is read and controller is run using a driver in files @c encoder_driver.h and
 *           @c encoder_driver.cpp.
 */

class task_control : public TaskBase
{
    private:
        // No private variables or methods for this class

    protected:
        int32_t current_pos_1;
        int32_t current_pos_2;
        int32_t ref_pos_1;
        int32_t ref_pos_2;
```

```
        const double KP_1 = .5;
        const double KI_1 = .05;
        const double KP_2 = 1;
        const double KI_2 = .01;
        int8_t KD;
        int32_t error_old_1;
        int32_t error_old_2;
        int32_t error_1;
        int32_t error_2;
        double KP_out_1;
        double KP_out_2;
        double KI_out_1;
        double KI_out_2;
        double KD_out_1;
        double KD_out_2;
        double speed_out_1;
        double speed_out_2;
        int8_t dead_zone;
        uint16_t hinge_limit;
        uint8_t count;

    public:
        // This constructor creates a generic task of which many copies can be made
        task_control (const char*, unsigned portBASE_TYPE, size_t, emstream*);

        // This method is called by the RTOS once to run the task loop for ever and ever.
        void run (void);
};

// This operator prints the A/D converter (see file adc.cpp for details). It's not
// a part of class adc, but it operates on objects of class adc
// emstream& operator << (emstream&, adc&);
emstream& operator << (emstream&, task_control&);

#endif // _TASK_CONTROL_H_
```

```
//*****************************************************************************
/** @file task_control.cpp
 *    This file contains the code for a task class which controls motor position
 *    of an electric motor.
 *
 *  Revisions:
 *    @li 02-17-2015 CAL, TT, JH Original file was a LED brightness control created
 *                    by JRR. This file has been modified to contain motor control
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************

#include "textqueue.h"                      // Header for text queue class
#include "task_control.h"                   // Header for this task
#include "shares.h"                         // Shared inter-task communications
#include <math.h>                           // Includes math library

#define brake_1 0                           // These defines help make the code more
#define free_1  1                           // readable. Enumeration data types were
#define power_1 2                           // not implemented for the shared data item
#define brake_2 3                           // p_mode, thus defining the integers in this
#define free_2  4                           // way makes the code easier to read and
#define power_2 5                           // understand.

//-----------------------------------------------------------------------------
/** This constructor creates a task which reads input from an encoder and controls the
 *  encoder using input from @c task_user. The main job of this constructor is to call the
 *  constructor of parent class (\c frt_task ).
 *  @param a_name A character string which will be the name of this task
 *  @param a_priority The priority at which this task will initially run (default: 0)
 *  @param a_stack_size The size of this task's stack in bytes
 *                      (default: configMINIMAL_STACK_SIZE)
 *  @param p_ser_dev Pointer to a serial device (port, radio, SD card, etc.) which can
 *                   be used by this task to communicate (default: NULL)
 */

task_control::task_control (const char* a_name, unsigned portBASE_TYPE a_priority,
                            size_t a_stack_size, emstream* p_ser_dev)
                            : TaskBase (a_name, a_priority, a_stack_size, p_ser_dev)
{

    // Nothing is done in the body of this constructor. All the work is done in the
    // call to the frt_task constructor on the line just above this one
}


//-----------------------------------------------------------------------------
/** This method is called once by the RTOS scheduler. It constructs the encoder
 *  to run using external interrupts. Each time around the for (;;) loop, the encoder
 *  is updated with the latest shared variables from the motor and/or task_user.
 */
```

```
void task_control::run (void)
{
    // Make a variable which will hold times to use for precise task scheduling
    TickType_t previousTicks = xTaskGetTickCount ();

    dead_zone = 20;
    hinge_limit = 1100;

    // This is the task loop for the control task. This loop runs until the
    // power is turned off or something equally dramatic occurs
    for (;;)
    {
        ref_pos_1 = p_position_1->get();
        ref_pos_2 = p_position_2->get();

        current_pos_1 = p_encoder_cntr_1->get();
        current_pos_2 = p_encoder_cntr_2->get();

        error_1 = ref_pos_1 - current_pos_1;
        error_2 = ref_pos_2 - current_pos_2;

        KP_out_1 = error_1*KP_1;
        KP_out_2 = error_2*KP_2;

        KI_out_1 = ((error_old_1 + error_1) * KI_1);
        KI_out_2 = ((error_old_2 + error_2) * KI_2);

        speed_out_1 = (KP_out_1 + KI_out_1 + KD_out_1);
        speed_out_2 = (KP_out_2 + KI_out_2 + KD_out_2);

//      MOTOR 1:
        // Brakes the motor if close to final position
        if ((error_1<=10 && error_1>=-10) || (speed_out_1==0))
        {
            p_mode -> put(brake_1);
            p_pos_done_1 -> put(true);
        }

        // Brakes the motor if close to hinge limit
        else if ((speed_out_1 > 1) && (current_pos_1 >= hinge_limit))
        {
            p_mode -> put(brake_1);
        }
        // Brakes the motor if going past zero
        else if ((speed_out_1 < -1) && (current_pos_1 <= 0))
        {
            p_mode -> put(brake_1);
        }

        // Positive speed cap
        else if (speed_out_1 > 300)
        {
            speed_out_1 =300;
            p_mode -> put(power_1);
        }

        // Motor will not spin unless power is greater than that defined by
        // the dead_zone variable
        else if (speed_out_1 < dead_zone && speed_out_1>0)
        {
            speed_out_1=dead_zone;
            p_mode -> put(power_1);
        }

        else if (speed_out_1 > -dead_zone && speed_out_1<0)
```

```
        {
            speed_out_1=-dead_zone;
            p_mode -> put(power_1);
        }

        // Negative speed cap
        else if(speed_out_1 < -300)
        {
            speed_out_1 = -300;
            p_mode -> put(power_1);
        }

        else
        {
            p_mode -> put(power_1);
        }

        error_old_1 = error_1;
        p_share_1 -> put(speed_out_1);
        p_position_1 -> put(ref_pos_1);

        //Delay control loop between motor 1 and motor 2
        delay_from_for_ms (previousTicks, 30);

//      MOTOR 2:
        // Brakes the motor if close to final position
        if((error_2<=30 && error_2>=-30) /*|| (speed_out_2==0)*/)
        {
            p_mode -> put(brake_2);
            p_pos_done_2 -> put(true);
        }

        // Positive speed cap
        else if(speed_out_2 > 300)
        {
            speed_out_2 =300;
            p_mode -> put(power_2);
        }

        // Motor will not spin unless power is greater than that defined by
        // the dead_zone variable
        else if ((speed_out_2 < dead_zone) && (speed_out_2>0))
        {
            speed_out_2=dead_zone;
            p_mode -> put(power_2);
        }

        else if ((speed_out_2 > -dead_zone) && (speed_out_2<0))
        {
            speed_out_2=-dead_zone;
            p_mode -> put(power_2);
        }

        // Negative speed cap
        else if(speed_out_2 < -300)
        {
            speed_out_2 = -300;
            p_mode -> put(power_2);
        }

        else
        {
            p_mode -> put(power_2);
        }
```

```
        error_old_2 = error_2;
        p_share_2 -> put(speed_out_2);
        p_position_2 -> put(ref_pos_2);

        // Outputs position to serial port for debugging.
        // Note: motors will not run without ARS being printed to the serial port for some reason
        *p_print_ser_queue  << "A: "<< current_pos_2 << endl;
        *p_print_ser_queue  << "R: "<< ref_pos_2 << endl;
        *p_print_ser_queue  << "S: "<< speed_out_2 << endl;

        // This is a method we use to cause a task to make one run through its task
        // loop every N milliseconds and let other tasks run at other times
        delay_from_for_ms (previousTicks, 30);
    }
}
```

```
//****************************************************************************************
/** @file shares.h
 *    This file contains extern declarations for queues and other inter-task data
 *    communication objects used in a ME405/507/FreeRTOS project.
 *
 *  Revisions:
 *    @li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    @li 10-05-2012 JRR Split into multiple files, one for each task plus a main one
 *    @li 10-29-2012 JRR Reorganized with global queue and shared data references
 *    @li 01-04-2014 JRR Re-reorganized, allocating shares with new now
 *    @li 02-09-2015 JH, TT, CL Added shared variables to be used in multiple tasks
 *        and ISRs
 *
 *  License:
 *      This file is copyright 2015 by JR Ridgely and released under the Lesser GNU
 *      Public License, version 2. It intended for educational use only, but its use
 *      is not limited thereto. */
/*      THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *      AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *      IMPLIED     WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *      ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *      LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *      TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *      OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *      CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *      OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *      OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//****************************************************************************************

// This define prevents this .h file from being included multiple times in a .cpp file
#ifndef _SHARES_H_
#define _SHARES_H_

//----------------------------------------------------------------------------------------
// Externs:  In this section, we declare variables and functions that are used in all
// (or at least two) of the files in the data acquisition project. Each of these items
// will also be declared exactly once, without the keyword 'extern', in one .cpp file
// as well as being declared extern here.

// This queue allows tasks to send characters to the user interface task for display.
extern TextQueue* p_print_ser_queue;

// This shared data item allows a value for motor 1 speed and direction to be shared between
// the task_user and task_motor. Task_user is the source of the data item, and task_motor
// is the sink that utilizes the data item.
extern TaskShare<int16_t>* p_share_1;

// This shared data item allows a value for motor 2 speed and direction to be shared between
// the task_user and task_motor. Task_user is the source of the data item, and task_motor
// is the sink that utilizes the data item.
extern TaskShare<int16_t>* p_share_2;

// This shared data item is used to set the motor mode: power, brake, freewheel for motors
// 1 and 2. Motor 1 uses integers 0 through 2 and motor 2 uses integers 3 through 5.
extern TaskShare<uint8_t>* p_mode;

// This shared data item is used to read the state of the encoder tick used for comparison
// in the ISR.
extern TaskShare<uint8_t>* p_state;

// This shared data item is used to hold the state of the last encoder tick of encoder 1 used for
// comparison in the ISR.
extern TaskShare<uint8_t>* p_state_old_1;

// This shared data item is used to hold the state of the last encoder tick of encoder 2 used for
```

```
// comparison in the ISR.
extern TaskShare<uint8_t>* p_state_old_2;

// This shared data item is used to hold the count for current number of errors. This number
// is incremented in the ISR when applicable and read using error_count method.
extern TaskShare<uint32_t>* p_error_cntr;

// This shared data item is used to hold the count for current position of encoder 1. This number
// is incremented or decremented based on comparsion of current and old states in the ISR and
// read using view_count method.
extern TaskShare<uint32_t>* p_encoder_cntr_1;

// This shared data item is used to hold the count for current position of encoder 2. This number
// is incremented or decremented based on comparsion of current and old states in the ISR and
// read using view_count method.
extern TaskShare<uint32_t>* p_encoder_cntr_2;

// This shared data is used to hold the location of channel A external interrupt pin.
extern TaskShare<uint8_t>* p_ext_pin_A;

// This shared data is used to hold the location of channel B external interrupt pin.
extern TaskShare<uint8_t>* p_ext_pin_B;

// This shared data is used to hold the location of channel C external interrupt pin.
extern TaskShare<uint8_t>* p_ext_pin_C;

// This shared data is used to hold the location of channel D external interrupt pin.
extern TaskShare<uint8_t>* p_ext_pin_D;

// This shared data item is used to hold the position sent to the control loop for motor 1
extern TaskShare<int16_t>* p_position_1;

// This shared data item is used to hold the position sent to the control loop for motor 2
extern TaskShare<int16_t>* p_position_2;

// This shared data item is used to signal task_trigger to pull the gun's trigger
extern TaskShare<bool>* fire_at_will;

// These are shared data items to read the IR sensors
extern TaskShare<uint16_t>* p_high_left;
extern TaskShare<uint16_t>* p_high_right;
extern TaskShare<uint16_t>* p_center;
extern TaskShare<uint16_t>* p_low_left;
extern TaskShare<uint16_t>* p_low_right;

// These shared data items are position done flags for the control loop
extern TaskShare<bool>* p_pos_done_1;
extern TaskShare<bool>* p_pos_done_2;


#endif // _SHARES_H_
```

```
//=============================================================================================
/** @file motor_driver.h
 *    This file contains a DC motor driver for the VNH3SP30 motor driver attached to the
 *    ME 405 board. This driver can control separate motors, causing them to either
 *    rotate clockwise, counterclockwise, brake, or freely rotate. The driver is
 *    hopefully thread safe in FreeRTOS due to the use of a mutex to prevent its use by
 *    multiple tasks at the same time. There is no protection from priority inversion,
 *    however, except for the priority elevation in the mutex.
 *
 *  Revisions:
 *    @li 01-15-2008 JRR Original (somewhat useful) file
 *    @li 10-11-2012 JRR Less original, more useful file with FreeRTOS mutex added
 *    @li 10-12-2012 JRR There was a bug in the mutex code, and it has been fixed
 *    @li 01-26-2015 CAL, TT, JH motor driver class implemented
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//=============================================================================================

// This define prevents this .H file from being included multiple times in a .CPP file
#ifndef MOTOR_DRIVER
#define MOTOR_DRIVER

#include "emstream.h"                       // Header for serial ports and devices
#include "FreeRTOS.h"                       // Header for the FreeRTOS RTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // Header for FreeRTOS queues
#include "semphr.h"                         // Header for FreeRTOS semaphores


//---------------------------------------------------------------------------------------------
/** @brief   This class will operate the VNH3SP30 motor driver on an AVR processor.
 *  @details The class contains a protected pointer to the serial port for outputs.
 *           It also contains pointers for motor driver PORTs, DDRs, and pin addresses.
 *           Public function set_power controls the motor speed based on a signed 16-bit
 *           input. Public function brake grounds both the motor leads so it will not
 *           spin. Public function freewheel sets the motor speed to zero, allowing it to
 *           rotate freely.
 */

class Motor
{
    protected:
        /// The Motor class uses this pointer to the serial port to say hello
        emstream* ptr_to_serial;

        // Pointers to motor driver PORTs, DDRs, and pin addresses
        volatile uint8_t* INa_PORT;
        volatile uint8_t* INa_DDR;
        uint8_t   INa_PIN;
        volatile uint8_t* INb_PORT;
        volatile uint8_t* INb_DDR;
        uint8_t   INb_PIN;
```

```
        volatile uint8_t* DIAG_PORT;
        volatile uint8_t* DIAG_DDR;
        uint8_t   DIAG_PIN;
        volatile uint8_t* PWM_PORT;
        volatile uint8_t* PWM_DDR;
        uint8_t   PWM_PIN;
        volatile uint16_t* PWM_OCR;

    public:
        // The constructor sets up the Motor driver for use. The "= NULL" part is a
        // default parameter, meaning that if that parameter isn't given on the line
        // where this constructor is called, the compiler will just fill in "NULL".
        // In this case that has the effect of turning off diagnostic printouts
        Motor (emstream* = NULL, volatile uint8_t* a_port = NULL, volatile uint8_t* a_d = NULL,
               uint8_t a_pin = 0, volatile uint8_t* b_port = NULL, volatile uint8_t* b_d = NULL,
               uint8_t b_pin = 0, volatile uint8_t* diag_po = NULL, volatile uint8_t* diag_d = NULL,
               uint8_t diag_pi = 0, volatile uint8_t* pwm_po = NULL, volatile uint8_t* pwm_d = NULL,
               uint8_t pwm_pi = 0, volatile uint16_t* pwm_oc = NULL);

        // The set_power function
        void set_power(int16_t speed);

        // The freewheel function
        void freewheel(void);

        // The brake function
        void brake(void);

}; // end of class Motor


// This operator prints the Motor diagnostics (see file Motor.cpp for details). It's not
// a part of class Motor, but it operates on objects of class Motor
emstream& operator << (emstream&, Motor&);

#endif // MOTOR_DRIVER
```

```
//*****************************************************************************
/** @file motor_driver.cpp
 *    This file contains a DC motor driver for the VNH3SP30 motor driver attached to the
 *    ME 405 board. This driver can control separate motors, causing them to either
 *    rotate clockwise, counterclockwise, brake, or freely rotate.
 *
 *  Revisions:
 *    @li 01-15-2008 JRR Original (somewhat useful) file
 *    @li 10-11-2012 JRR Less original, more useful file with FreeRTOS mutex added
 *    @li 10-12-2012 JRR There was a bug in the mutex code, and it has been fixed
 *    @li 01-26-2015 CAL, TT, JH motor driver class implemented
 *
 *  License:
 *    This file is copyright 2015 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************

#include <stdlib.h>                         // Include standard library header files
#include <avr/io.h>
#include <math.h>

#include "rs232int.h"                       // Include header for serial port class
#include "motor_driver.h"                   // Include header for the motor driver class

//---------------------------------------------------------------------------
/** \brief This constructor sets up a motor driver to work with the VNH3SP30 motor
 *  driver.
 *  \details The motor driver is made ready so that when a method such as
 *  @c set_power() is called, the corresponding registers on the VNH3SP30 can be
 *  correctly set. The constructor contains a protected pointer to the serial port for
 *  outputs. It also contains pointers for motor driver PORTs, DDRs, and pin addresses.
 *  Public function set_power controls the motor speed based on a signed 16-bit
 *  input. Public function brake grounds both the motor leads so it will not
 *  spin. Public function freewheel sets the motor speed to zero, allowing it to
 *  rotate freely.
 *  @param p_serial_port A pointer to the serial port which writes debugging info.
 *  @param a_port A pointer to the port corresponding to VNH3SP30 pin INa
 *  @param a_d A pointer to the Data Direction Register for VNH3SP30 pin INa
 *  @param a_pin An integer for the pin/bit location corresponding to VNH3SP30 pin INa
 *  @param b_port A pointer to the port corresponding to VNH3SP30 pin INb
 *  @param b_d A pointer to the Data Direction Register for VNH3SP30 pin INb
 *  @param b_pin An integer for the pin/bit location corresponding to VNH3SP30 pin INb
 *  @param diag_po A pointer to the port corresponding to VNH3SP30 diagnostic pin:
 *                 DIAGA/B
 *  @param diag_d A pointer to the Data Direction Register for VNH3SP30 diagnostic pin:
 *                  DIAGA/B
 *  @param diag_pi An integer for the pin/bit location corresponding to VNH3SP30
 *                 diagnostic pin: DIAGA/B
 *  @param pwm_po A pointer to the port corresponding to VNH3SP30 PWM pin
 *  @param pwm_d A pointer to the Data Direction Register for VNH3SP30 pin INb
 *  @param pwm_pi An integer for the pin/bit location corresponding to VNH3SP30 pin INa
 *  @param pwm_oc A pointer to the Output Compare Register for the PWM
 */
```

```
Motor:: Motor (emstream* p_serial_port, volatile uint8_t* a_port, volatile uint8_t* a_d,
               uint8_t a_pin, volatile uint8_t* b_port, volatile uint8_t* b_d,
               uint8_t b_pin, volatile uint8_t* diag_po, volatile uint8_t* diag_d,
               uint8_t diag_pi, volatile uint8_t* pwm_po, volatile uint8_t* pwm_d,
               uint8_t pwm_pi, volatile uint16_t* pwm_oc)
{
        ptr_to_serial = p_serial_port;

        // Take constructor arguments and save them in corresponding protected variables
        // for the motor object
        INa_PORT = a_port;
        INa_DDR = a_d;
        INa_PIN = a_pin;
        INb_PORT = b_port;
        INb_DDR = b_d;
        INb_PIN = b_pin;
        DIAG_PORT = diag_po;
        DIAG_DDR = diag_d;
        DIAG_PIN = diag_pi;
        PWM_PORT = pwm_po;
        PWM_DDR = pwm_d;
        PWM_PIN = pwm_pi;
        PWM_OCR = pwm_oc;

        // Initialize port for PWM as an output and as high
        *PWM_PORT |= (1 << PWM_PIN);
        *PWM_DDR |= (1 << PWM_PIN);

        // Initialize port for the VNH3SP30 motor driver diagnostic pins, the port is
        // an input to the AVR and the pullup resistor is enabled
        *DIAG_PORT |= (1 << DIAG_PIN);
        *DIAG_DDR &= ~(1 << DIAG_PIN);

        // Initialize the Data Direction Registers for INa and INb pins as outputs
        *INa_DDR |= (1 << INa_PIN);
        *INb_DDR |= (1 << INb_PIN);

        // Motor debugging message
        DBG (ptr_to_serial, "Motor constructor OK" << endl);
}

//-------------------------------------------------------------------------------------
/** @brief   Takes a 16 bit signed input and sets the motor torque
 *  @details A positive input speed will spin the motor clockwise by setting INa to
 *           high and set the PWM Output Compare Register to the input speed. A negative
 *           input speed will spin the motor counterclockwise by setting INb to high and
 *           set the PWM Output Compare Register to the input speed.
 *  @param   speed A value used to set the PWM duty cycle
 */

void Motor::set_power(int16_t speed)
{
    //set motor to spin clockwise, by setting INa to high and set PWM period to input speed
    if (speed > 0)
    {
        *INa_PORT |= (1 << INa_PIN);
        *INb_PORT &= ~(1 <<INb_PIN);

        *PWM_OCR = abs(speed);
    }
    //set motor to spin counterclockwise, by setting INb to high and set PWM period to input speed
    else
    {
        *INb_PORT |= (1 << INb_PIN);
        *INa_PORT &= ~(1 << INa_PIN);
```

```
        *PWM_OCR = abs(speed);
    }
}

//-------------------------------------------------------------------------------------
/** @brief   This method toggles motor freewheeling
 *  @details By setting the PWM Output Compare Register to zero, the motor will spin
 *           freely.
 */

void Motor::freewheel(void)
{
    *PWM_OCR = 0;
}

//-------------------------------------------------------------------------------------
/** @brief   Sets the motor to a braked state
 *  @details Grounds both the INa and INb pins so that the motor is braked. Sets the PWM
 *           for maximum braking power.
 */

void Motor::brake(void)
{
    *INa_PORT &= ~(1 << INa_PIN);
    *INb_PORT &= ~(1 << INb_PIN);

    *PWM_OCR = 255;
}

//-------------------------------------------------------------------------------------
/** \brief   This overloaded operator "prints the motor driver"
 *  \details This prints the motor driver control registers PORTC, PORTD, PORTB, DDRC
 *           DDRD, DDRB, OCR1A, and OCR1B. This is useful for debugging purposes.
 *  @param   serpt Reference to a serial port to which the printout will be printed
 *  @param   vroom Reference to the motor driver which is being printed
 *  @return  A reference to the same serial device on which we write information.
 *           This is used to string together things to write with @c << operators
 */

emstream& operator << (emstream& serpt, Motor& vroom)
{
//  These messages were used for debugging purposes, and are not printed during normal
//  operation.

//  serpt << PMS ("PORTC: ") <<bin << PORTC <<endl;
//  serpt << PMS ("DDRC:  ") <<bin << DDRC <<endl;
//  serpt << PMS ("PORTD: ") <<bin << PORTD <<endl;
//  serpt << PMS ("DDRD:  ") <<bin << DDRD <<endl;
//  serpt << PMS ("PORTB: ") <<bin << PORTB <<endl;
//  serpt << PMS ("DDRB:  ") <<bin << DDRB <<endl;
//  serpt << PMS ("PWM 1: ") <<bin << OCR1A <<endl;
//  serpt << PMS ("PWM 2: ") <<bin << OCR1B <<endl;
    return (serpt);
}
```

```
//****************************************************************************
/** \file main.cpp
 *    This file contains the main() code for a program which runs the ME405 board for
 *    ME405 Final Project. This program uses an A/D converter to convert an analog signal
 *    into an LED brightness via pulse width modulation. It also uses a motor driver to
 *    independently control separate motors connected to the VNH3P30 motor drivers
 *    of the ME 405 board.
 *
 *  Revisions:
 *    \li 09-30-2012 JRR Original file was a one-file demonstration with two tasks
 *    \li 10-05-2012 JRR Split into multiple files, one for each task plus a main one
 *    \li 10-30-2012 JRR A hopefully somewhat stable version with global queue
 *                       pointers and the new operator used for most memory allocation
 *    \li 11-04-2012 JRR FreeRTOS Swoop demo program changed to a sweet test suite
 *    \li 01-05-2012 JRR Program reconfigured as ME405 Lab 1 starting point
 *    \li 03-28-2014 JRR Pointers to shared variables and queues changed to references
 *    @li 01-04-2015 JRR Names of share & queue classes changed; allocated with new now
 *    @li 01-26-2015 CAL, TT, JH Updated comments and lab name
 *
 *  License:
 *      This file is copyright 2015 by JR Ridgely and released under the Lesser GNU
 *      Public License, version 2. It intended for educational use only, but its use
 *      is not limited thereto. */
/*      THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *      AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *      IMPLIED     WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *      ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *      LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *      TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *      OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *      CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *      OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *      OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//****************************************************************************


#include <stdlib.h>                     // Prototype declarations for I/O functions
#include <avr/io.h>                     // Port I/O for SFR's
#include <avr/wdt.h>                    // Watchdog timer header
#include <string.h>                     // Functions for C string handling

#include "FreeRTOS.h"                   // Primary header for FreeRTOS
#include "task.h"                       // Header for FreeRTOS task functions
#include "queue.h"                      // FreeRTOS inter-task communication queues
#include "croutine.h"                   // Header for co-routines and such

#include "rs232int.h"                   // ME405/507 library for serial comm.
#include "time_stamp.h"                 // Class to implement a microsecond timer
#include "taskbase.h"                   // Header of wrapper for FreeRTOS tasks
#include "textqueue.h"                  // Wrapper for FreeRTOS character queues
#include "taskqueue.h"                  // Header of wrapper for FreeRTOS queues
#include "taskshare.h"                  // Header for thread-safe shared data
#include "shares.h"                     // Global ('extern') queue declarations
#include "task_motor.h"                 // Header for the data acquisition task
#include "task_user.h"                  // Header for user interface task
#include "encoder_driver.h"             // Header for the encoder driver
#include "task_encoder.h"               // Header for the encoder task
#include "task_control.h"               // Header for the controller task
#include "task_sensor.h"                    // Header for the scan task
#include "task_trigger.h"               // Header for the trigger task
#include "task_position.h"              // Header for the position task

// Declare the queues which are used by tasks to communicate with each other here.
// Each queue must also be declared 'extern' in a header file which will be read
// by every task that needs to use that queue. The format for all queues except
```

```
// the serial text printing queue is 'frt_queue<type> name (size)', where 'type'
// is the type of data in the queue and 'size' is the number of items (not neces-
// sarily bytes) which the queue can hold

/** This is a print queue, descended from \c emstream so that things can be printed
 *  into the queue using the "<<" operator and they'll come out the other end as a
 *  stream of characters. It's used by tasks that send things to the user interface
 *  task to be printed.
 */
TextQueue* p_print_ser_queue;

// This shared data item allows a value for motor speed and direction to be shared between
// the task_user and task_motor for each motor 1 and 2. Task_user is the source of the data item, and
task_motor
// is the sink that utilizes the data item.
TaskShare<int16_t>* p_share_1;
TaskShare<int16_t>* p_share_2;

// This shared data item is used to set the motor mode: power, brake, freewheel for motors
// 1 and 2. Motor 1 uses integers 0 through 2 and motor 2 uses integers 3 through 5.
TaskShare<uint8_t>* p_mode;

// This shared data item is used to read the state of the encoder tick used for comparison
// in the ISR.
TaskShare<uint8_t>* p_state;

// This shared data item is used to hold the state of the last encoder tick from encoder 1 used for
// comparison in the ISR.
TaskShare<uint8_t>* p_state_old_1;

// This shared data item is used to hold the state of the last encoder tick from encoder 2 used for
// comparison in the ISR.
TaskShare<uint8_t>* p_state_old_2;

// This shared data item is used to hold the count for current number of errors. This number
// is incremented in the ISR when applicable and read using error_count method.
TaskShare<uint32_t>* p_error_cntr;

// This shared data item is used to hold the count for current position of encoder 1. This number
// is incremented or decremented based on comparsion of current and old states in the ISR and
// read using view_count method.
TaskShare<uint32_t>* p_encoder_cntr_1;

// This shared data item is used to hold the count for current position of encoder 2. This number
// is incremented or decremented based on comparsion of current and old states in the ISR and
// read using view_count method.
TaskShare<uint32_t>* p_encoder_cntr_2;

// This shared data is used to hold the location of channel A external interrupt pin.
TaskShare<uint8_t>* p_ext_pin_A;

// This shared data is used to hold the location of channel B external interrupt pin.
TaskShare<uint8_t>* p_ext_pin_B;

// This shared data is used to hold the location of channel C external interrupt pin.
TaskShare<uint8_t>* p_ext_pin_C;

// This shared data is used to hold the location of channel D external interrupt pin.
TaskShare<uint8_t>* p_ext_pin_D;

// This shared data item is used to hold the position of motor 1 sent to the control loop
TaskShare<int16_t>* p_position_1;

// This shared data item is used to hold the position of motor 2 sent to the control loop
TaskShare<int16_t>* p_position_2;
```

```
// This shared data item is used to signal task_trigger to pull the gun's trigger
TaskShare<bool>* fire_at_will;

// These are shared data items to read the IR sensors
TaskShare<uint16_t>* p_high_left;
TaskShare<uint16_t>* p_high_right;
TaskShare<uint16_t>* p_center;
TaskShare<uint16_t>* p_low_left;
TaskShare<uint16_t>* p_low_right;

// These are shared data items that signal when a particular motor's control loop has
// reached the desired value
TaskShare <bool>* p_pos_done_1;
TaskShare <bool>* p_pos_done_2;
//=================================================================================
/** The main function sets up the RTOS.  Some test tasks are created. Then the
 *  scheduler is started up; the scheduler runs until power is turned off or there's a
 *  reset.
 *  @return This is a real-time microcontroller program which doesn't return. Ever.
 */

int main (void)
{
    // Disable the watchdog timer unless it's needed later. This is important because
    // sometimes the watchdog timer may have been left on...and it tends to stay on
    MCUSR = 0;
    wdt_disable ();

    // Configure a serial port which can be used by a task to print debugging infor-
    // mation, or to allow user interaction, or for whatever use is appropriate.  The
    // serial port will be used by the user interface task after setup is complete and
    // the task scheduler has been started by the function vTaskStartScheduler()
    rs232* p_ser_port = new rs232 (9600, 1);
    *p_ser_port << clrscr << PMS ("ME405 Term Project Tasks") << endl;

    // Create the queues and other shared data items here
    p_print_ser_queue = new TextQueue (32, "Print", p_ser_port, 10);

    // Create shared variables for motor control
    p_share_1 = new TaskShare<int16_t> ("Speed_1");
    p_share_2 = new TaskShare<int16_t> ("Speed_2");
    p_mode = new TaskShare<uint8_t> ("Mode");
    p_state= new TaskShare<uint8_t> ("State");

    // Create shared variables for encoder states and positions
    p_state_old_1= new TaskShare<uint8_t> ("StateOld_1");
    p_state_old_2= new TaskShare<uint8_t> ("StateOld_2");
    p_position_1= new TaskShare<int16_t> ("Pos_1");
    p_position_2= new TaskShare<int16_t> ("Pos_2");

    // Create shared variables for encoder counters
    p_encoder_cntr_1= new TaskShare<uint32_t> ("EncoderCntr_1");
    p_encoder_cntr_2= new TaskShare<uint32_t> ("EncoderCntr_2");
    p_error_cntr= new TaskShare<uint32_t> ("ErrorCntr");

    // Create shared variables for the encoder external interrupt pins
    p_ext_pin_A= new TaskShare<uint8_t> ("ExtPinA");
    p_ext_pin_B= new TaskShare<uint8_t> ("ExtPinB");
    p_ext_pin_C= new TaskShare<uint8_t> ("ExtPinC");
    p_ext_pin_D= new TaskShare<uint8_t> ("ExtPinD");

    // Create shared variable for the trigger
    fire_at_will = new TaskShare<bool> ("Shoot_em_up");
```

```
//  Create shared variable for the phototransistor sensor readings
    p_high_left= new TaskShare<uint16_t> ("P_high_L");
    p_high_right= new TaskShare<uint16_t> ("P_high_R");
    p_center= new TaskShare<uint16_t> ("P_center");
    p_low_left= new TaskShare<uint16_t> ("P_low_L");
    p_low_right= new TaskShare<uint16_t> ("P_low_R");

    // Create shared variables for signaling when a desired position has been reached
    p_pos_done_1 = new TaskShare <bool> ("Pos_done_1");
    p_pos_done_2 = new TaskShare <bool> ("Pos_done_2");

    // The user interface is at low priority; it is only used to print debugging messages
    // and restart the microcontroller in this application
    new task_user ("UserInt", task_priority (0), 260, p_ser_port);

    // Create a task which outputs to a motor driver
    new task_motor ("Motor", task_priority (2), 280, p_ser_port);

    // Create a task which monitors the activity of the optical encoders
    new task_encoder ("Encoder", task_priority (1), 280, p_ser_port);

    // Create a task which utilized the motor and encoder tasks to provide closed-loop
    // control of a motor
    new task_control ("Controller", task_priority (3), 280, p_ser_port);

    // Create a task to scan a bank of phototransistors using an A/D converter
    new task_sensor ("Sensor", task_priority (1), 280, p_ser_port);

    // Create a task to pull the trigger
    new task_trigger ("Trigger", task_priority (0), 280, p_ser_port);

    // Create a task to determine the current motors' positions
    new task_position ("Position", task_priority (4), 280, p_ser_port);

    // Here's where the RTOS scheduler is started up. It should never exit as long as
    // power is on and the microcontroller isn't rebooted
    vTaskStartScheduler ();
}
```

```
//=============================================================================================
/** @file encoder_driver.h
 *    This file contains a driver for an incremental optical encoder.This driver tracks
 *    the encoder count and the number of errors in reading during operation. The driver
 *    is hopefully thread safe in FreeRTOS due to the use of a mutex to prevent its use
 *    by multiple tasks at the same time. There is no protection from priority inversion,
 *    however, except for the priority elevation in the mutex.
 *
 *  Revisions:
 *    @li 02-09-2015 CAL, TT, JH encoder driver class implemented. Original file was
 *    a LED brightness control created by JRR
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//=============================================================================================

// This define prevents this .H file from being included multiple times in a .CPP file
#ifndef ENCODER_DRIVER
#define ENCODER_DRIVER

#include "emstream.h"                       // Header for serial ports and devices
#include "FreeRTOS.h"                       // Header for the FreeRTOS RTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // Header for FreeRTOS queues
#include "semphr.h"                         // Header for FreeRTOS semaphores
#include "taskshare.h"                      // Header for thread-safe shared data
#include "textqueue.h"                      // Header of wrapper for FreeRTOS queues
#include "shares.h"                         // Shared inter-task communications

//---------------------------------------------------------------------------------------------
/** @brief   This class will read an encoder connected to the AVR processor by any of
 *           the external input pins 4 through 7.
 *  @details The class contains a protected pointer to the serial port for outputs.
 *           It also contains pointers for encoder PORTs, DDRs, and pin addresses.
 *           Public function error_count displays the number of errors accumulated.
 *           Public function clear_count zeros the encoder count. Public function
 *           view_count displays the current encoder count.  Public function set_count
 *           sets the encoder count according to a user input.
 */

class Encoder
{
    protected:
        /// The Encoder class uses this pointer to the serial port to say hello
        emstream* ptr_to_serial;

        // Pointers to encoder PORTs, DDRs, and pin addresses
        volatile uint8_t* INTERRUPT_PORT;
        volatile uint8_t* INTERRUPT_DDR;
        uint8_t   INTERRUPT_PIN_0_A;
        uint8_t   INTERRUPT_PIN_1_A;
        uint8_t   INTERRUPT_PIN_0_B;
        uint8_t   INTERRUPT_PIN_1_B;
```

```
        uint8_t   EXT_PIN_NUMBER_A;
        uint8_t   EXT_PIN_NUMBER_B;

        // Storage variables for encoder class
        uint32_t  ENCODER_COUNT;
        uint32_t  ERROR_COUNT;
        uint8_t   STATE;
        uint8_t   STATE_OLD;

    public:
        // The constructor sets up the Encoder driver for use. The "= NULL" part is a
        // default parameter, meaning that if that parameter isn't given on the line
        // where this constructor is called, the compiler will just fill in "NULL".
        // In this case that has the effect of turning off diagnostic printouts
        Encoder (emstream* = NULL, volatile uint8_t* i_port = NULL,
                 volatile uint8_t* i_ddr = NULL, volatile uint8_t* i_port_in = NULL,
                 uint8_t i_pin_0_a = 0, uint8_t i_pin_1_a = 0, uint8_t e_pin_a = 0,
                 uint8_t i_pin_0_b = 0, uint8_t i_pin_1_b = 0, uint8_t e_pin_b = 0);

        // Method that counts errors
        uint16_t error_count(uint8_t enc_num);

        // Method that clears the encoder count
        void clear_count(uint8_t enc_num);

        // Method that allows for encoder count viewing
        void view_count(uint8_t enc_num);

        // Method that sets encoder count
        void set_count(uint8_t enc_num, uint32_t NEW_COUNT);

}; // end of class Encoder

// This operator prints the Encoder diagnostics (see file Encoder.cpp for details). It's not
// a part of class Encoder, but it operates on objects of class Encoder
emstream& operator << (emstream&, Encoder&);

#endif // ENCODER_DRIVER
```

```
//*********************************************************************************
/** @file encoder_driver.cpp
 *    This file contains a driver for an incremental optical encoder.This driver tracks
 *    the encoder count and the number of errors in reading during operation. The
 *    encoder ISR code is also located within this file.
 *
 *  Revisions:
 *    @li 02-09-2015 CAL, TT, JH encoder driver class implemented. Original file was
 *    a LED brightness control created by JRR
 *
 *  License:
 *    This file is copyright 2015 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*********************************************************************************

#include <stdlib.h>                         // Include standard library header files
#include <avr/io.h>
#include <avr/interrupt.h>
#include <math.h>

#include "rs232int.h"                       // Include header for serial port class
#include "encoder_driver.h"                 // Include header for the encoder driver class

#include "shares.h"                         // Shared inter-task communications

//-------------------------------------------------------------------------------
/** \brief This constructor sets up an instance of the class encoder_driver
 *  \details The encoder is made ready so that when a method such as @c view_count() is
 *  called, the corresponding registers can be correctly set. The constructor contains
 *  a protected pointer to the serial port for outputs. The constructor contains a
 *  protected pointer to the serial port for outputs. It also sets pointers for
 *  encoder PORTs, DDRs, and pin addresses. It generates an interrupt to run on any
 *  logic change on the external interrupt input pins.
 *  @param p_serial_port A pointer to the serial port which writes debugging info.
 *  @param i_port A pointer to the corresponding PORT for the encoder.
 *  @param i_ddr  A pointer to the corresponding DDR for the encoder.
 *  @param i_pin_0_a The pin number for EICRB register bit 0 of external pin A
 *                   the encoder is wired to
 *  @param i_pin_1_a The pin number for EICRB register bit 1 of external pin A
 *                   the encoder is wired to
 *  @param e_pin_a The pin number for the external input pin A the encoder is wired to.
 *  @param i_pin_0_b A pointer to the EICRB register bit 0 of the external pin B
 *                   the encoder is wired to
 *  @param i_pin_1_b The pin number for EICRB register bit 0 of external pin B
 *                   the encoder is wired to
 *  @param e_pin_b The pin number for the external input pin A the encoder is wired to.
 */

Encoder::Encoder (emstream* p_serial_port, volatile uint8_t* i_port,  volatile uint8_t* i_ddr,
                  volatile uint8_t* i_port_in, uint8_t i_pin_0_a, uint8_t i_pin_1_a,
                  uint8_t e_pin_a, uint8_t i_pin_0_b, uint8_t i_pin_1_b, uint8_t e_pin_b)
{
        ptr_to_serial = p_serial_port;
```

```
        // Take constructor arguments and save them in corresponding protected variables
        // for the motor object
        INTERRUPT_PORT = i_port;
        INTERRUPT_DDR = i_ddr;
        INTERRUPT_PIN_0_A = i_pin_0_a;
        INTERRUPT_PIN_1_A = i_pin_1_a;
        INTERRUPT_PIN_0_B = i_pin_0_b;
        INTERRUPT_PIN_1_B = i_pin_1_b;
        EXT_PIN_NUMBER_A = e_pin_a;
        EXT_PIN_NUMBER_B = e_pin_b;

        // Since two encoders are being used for Jankbot, this snippet of code ensures
        // that the correct set of external pins are configured for each instance of class
        // encoder_driver
        if(e_pin_b < 6)
        {
        p_ext_pin_A -> put (e_pin_a);
        p_ext_pin_B -> put (e_pin_b);
        }

        else
        {
        p_ext_pin_C -> put (e_pin_a);
        p_ext_pin_D -> put (e_pin_b);
        }

        // Initialize external interrupt pin pullup resistors
        *INTERRUPT_PORT |= (1 << EXT_PIN_NUMBER_A) | (1 << EXT_PIN_NUMBER_B);

        // Initialize external interrupt pins as inputs into AVR
        *INTERRUPT_DDR &= ~(1 << EXT_PIN_NUMBER_A) & ~(1 << EXT_PIN_NUMBER_B);

        // Enable interrupt triggering for external pins
        EIMSK |= (1 << EXT_PIN_NUMBER_A) | (1 << EXT_PIN_NUMBER_B);

        // Any logic change in input pins generate an interrupt
        EICRB |= (1 << INTERRUPT_PIN_0_A) | (1 << INTERRUPT_PIN_0_B);

        // Initializes the encoder's error count
        ERROR_COUNT = 0;
        p_error_cntr -> put(ERROR_COUNT);

        // Encoder debugging message
        DBG (ptr_to_serial, "Encoder constructor OK" << endl);
}

//-------------------------------------------------------------------------------------
/** @brief   Displays the current error count
 *  @details By using the shared variable p_error_cntr the error count is pulled and
 *           displayed through a debug message sent to the serial port.
 */

uint16_t Encoder::error_count(uint8_t enc_num)
{
    if (enc_num == 1)
    {
        ERROR_COUNT = p_error_cntr ->get();
    }

    else if (enc_num == 2)
    {
        ERROR_COUNT = p_error_cntr ->get();
    }

    // This message was used for debugging purposes, but is unused in this iteration
```

```
    // of the code.
//  DBG (ptr_to_serial, "Can has error? " << ERROR_COUNT << endl);
}

//-------------------------------------------------------------------------------
/** @brief   Sets the encoder count to 0
 *  @details Sets the share variable p_encoder_cntr to 0
 */

void Encoder::clear_count(uint8_t enc_num)
{
    ENCODER_COUNT = 0;
    if (enc_num == 1)
    {
        p_encoder_cntr_1 -> put(ENCODER_COUNT);
    }

    else if (enc_num == 2)
    {
        p_encoder_cntr_2 -> put(ENCODER_COUNT);
    }
}

//-------------------------------------------------------------------------------
/** @brief   Displays the current encoder count
 *  @details Collects data from the shared variable p_encoder_cntr and displays it
 *           through a debug message sent to the serial port.
 */

void Encoder::view_count(uint8_t enc_num)
{
    if (enc_num == 1)
    {
        ENCODER_COUNT = p_encoder_cntr_1 ->get();
    }

    else if (enc_num == 2)
    {
        ENCODER_COUNT = p_encoder_cntr_2 ->get();
    }
    // This message was used for debugging purposes, but is unused in this iteration
    // of the code.
//  DBG (ptr_to_serial, "Counting with Dora:" << (int32_t)ENCODER_COUNT << endl);
}

//-------------------------------------------------------------------------------
/** @brief   Sets the encoder count to a specific input
 *  @details Sets the shared variable p_encoder_cntr to the NEW_COUNT
 *  @param   NEW_COUNT A uint32_t variable containing the new encoder count number
 */

void Encoder::set_count(uint8_t enc_num, uint32_t NEW_COUNT)
{
    if (enc_num == 1)
    {
        p_encoder_cntr_1->put(NEW_COUNT);
    }

    else if (enc_num == 2)
    {
        p_encoder_cntr_2->put(NEW_COUNT);
    }
}

//-------------------------------------------------------------------------------
```

```
/** \brief   This overloaded operator "prints the encoder"
 *  \details This prints the relevant encoder registers PORTE. It is useful for
 *           debugging purposes but is not utilized in this build of the driver.
 *  @param   serpt Reference to a serial port to which the printout will be printed
 *  @param   vroom Reference to the motor driver which is being printed
 *  @return  A reference to the same serial device on which we write information.
 *           This is used to string together things to write with @c << operators
 */

emstream& operator << (emstream& serpt, Encoder& vroom)
{
//  These messages were used for debugging purposes, and are not printed during normal
//  operation.

//  serpt << PMS ("PORTE: ") <<bin << PORTE <<endl;

    return (serpt);
}

//-------------------------------------------------------------------------------------
/** This interrupt service routine runs whenever an external input pin changes either
 *  from low to high or from high to low. The ISR stores the encoder count and
 *  determines the direction the encoder is running by checking the current state with
 *  the previous state. It also checks that the encoder does not skip a count, if so it
 *  increments the error counter.
 */

// Encoder 1:
ISR (INT4_vect)
{
    // Declaration of state variables and input of shared values
    uint8_t STATE = PINE;
    uint8_t STATE_OLD= p_state_old_1 -> ISR_get();
    uint8_t EXT_PIN_NUMBER_A= p_ext_pin_A-> ISR_get();
    uint8_t EXT_PIN_NUMBER_B= p_ext_pin_B-> ISR_get();
    uint32_t ENCODER_COUNT= p_encoder_cntr_1-> ISR_get();
    uint32_t ERROR_COUNT= p_error_cntr-> ISR_get();

    // For Channel A and Channel B state: 00
    if (!(STATE & (1 <<EXT_PIN_NUMBER_A)) && !(STATE & (1 <<EXT_PIN_NUMBER_B)))
    {
        // If previous state was 01, increment ENCODER_COUNT in positive direction
        if (!(STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && (STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT++;
        }

        // If previous state was 10, increment ENCODER_COUNT in negative direction
        else if ((STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && !(STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT--;
        }

        // If previous state was neither 01 or 10, increment ERROR_COUNT
        else
        {
            ERROR_COUNT++;
        }
    }

    // 01
    if (!(STATE & (1 <<EXT_PIN_NUMBER_A)) && (STATE & (1 <<EXT_PIN_NUMBER_B)))
    {
        // If previous state was 11, increment ENCODER_COUNT in positive direction
        if ((STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && (STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
```

```
        {
            ENCODER_COUNT++;
        }

        // If previous state was 00, increment ENCODER_COUNT in negative direction
        else if (!(STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && !(STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT--;
        }

        // If previous state was neither 11 or 00, increment ERROR_COUNT
        else
        {
            ERROR_COUNT++;
        }
    }

    // 10
    if ((STATE & (1 <<EXT_PIN_NUMBER_A)) && !(STATE & (1 <<EXT_PIN_NUMBER_B)))
    {
        // If previous state was 00, increment ENCODER_COUNT in positive direction
        if (!(STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && !(STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT++;
        }

        // If previous state was 11, increment ENCODER_COUNT in negative direction
        else if ((STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && (STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT--;
        }

        // If previous state was neither 00 or 11, increment ERROR_COUNT
        else
        {
            ERROR_COUNT++;
        }
    }

    // 11
    if ((STATE & (1 <<EXT_PIN_NUMBER_A)) && (STATE & (1 <<EXT_PIN_NUMBER_B)))
    {
        // If previous state was 10, increment ENCODER_COUNT in positive direction
        if ((STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && !(STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT++;
        }

        // If previous state was 01, increment ENCODER_COUNT in negative direction
        else if (!(STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && (STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT--;
        }

        // If previous state was neither 10 or 01, increment ERROR_COUNT
        else
        {
            ERROR_COUNT++;
        }
    }

        p_encoder_cntr_1->ISR_put(ENCODER_COUNT);
        p_state_old_1->ISR_put(STATE);
        p_error_cntr->ISR_put(ERROR_COUNT);
}
```

```
// Alias external interrupt pin 5 to input pin 4 interrupt service routine
ISR_ALIAS(INT5_vect, INT4_vect);


// Encoder 2:
ISR (INT6_vect)
{
    // Declaration of state variables and input of shared values
    uint8_t STATE = PINE;
    uint8_t STATE_OLD= p_state_old_2 -> ISR_get();
    uint8_t EXT_PIN_NUMBER_A= p_ext_pin_C-> ISR_get();
    uint8_t EXT_PIN_NUMBER_B= p_ext_pin_D-> ISR_get();
    uint32_t ENCODER_COUNT= p_encoder_cntr_2-> ISR_get();
    uint32_t ERROR_COUNT= p_error_cntr-> ISR_get();

    // For Channel A and Channel B state: 00
    if (!(STATE & (1 <<EXT_PIN_NUMBER_A)) && !(STATE & (1 <<EXT_PIN_NUMBER_B)))
    {
        // If previous state was 01, increment ENCODER_COUNT in positive direction
        if (!(STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && (STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT++;
        }

        // If previous state was 10, increment ENCODER_COUNT in negative direction
        else if ((STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && !(STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT--;
        }

        // If previous state was neither 01 or 10, increment ERROR_COUNT
        else
        {
            ERROR_COUNT++;
        }
    }

    // 01
    if (!(STATE & (1 <<EXT_PIN_NUMBER_A)) && (STATE & (1 <<EXT_PIN_NUMBER_B)))
    {
        // If previous state was 11, increment ENCODER_COUNT in positive direction
        if ((STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && (STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT++;
        }

        // If previous state was 00, increment ENCODER_COUNT in negative direction
        else if (!(STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && !(STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT--;
        }

        // If previous state was neither 11 or 00, increment ERROR_COUNT
        else
        {
            ERROR_COUNT++;
        }
    }

    // 10
    if ((STATE & (1 <<EXT_PIN_NUMBER_A)) && !(STATE & (1 <<EXT_PIN_NUMBER_B)))
    {
        // If previous state was 00, increment ENCODER_COUNT in positive direction
        if (!(STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && !(STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
```

```
        {
            ENCODER_COUNT++;
        }

        // If previous state was 11, increment ENCODER_COUNT in negative direction
        else if ((STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && (STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT--;
        }

        // If previous state was neither 00 or 11, increment ERROR_COUNT
        else
        {
            ERROR_COUNT++;
        }
    }

    // 11
    if ((STATE & (1 <<EXT_PIN_NUMBER_A)) && (STATE & (1 <<EXT_PIN_NUMBER_B)))
    {
        // If previous state was 10, increment ENCODER_COUNT in positive direction
        if ((STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && !(STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT++;
        }

        // If previous state was 01, increment ENCODER_COUNT in negative direction
        else if (!(STATE_OLD & (1 <<EXT_PIN_NUMBER_A)) && (STATE_OLD & (1 <<EXT_PIN_NUMBER_B)))
        {
            ENCODER_COUNT--;
        }

        // If previous state was neither 10 or 01, increment ERROR_COUNT
        else
        {
            ERROR_COUNT++;
        }
    }

        p_encoder_cntr_2->ISR_put(ENCODER_COUNT);
        p_state_old_2->ISR_put(STATE);
        p_error_cntr->ISR_put(ERROR_COUNT);
}

// Alias external interrupt pin 7 to input pin 6 interrupt service routine
ISR_ALIAS(INT7_vect, INT6_vect);
```

```
//=====================================================================================
/** @file adc.h
 *    This file contains a very simple A/D converter driver. The driver is hopefully
 *    thread safe in FreeRTOS due to the use of a mutex to prevent its use by multiple
 *    tasks at the same time. There is no protection from priority inversion, however,
 *    except for the priority elevation in the mutex.
 *
 *  Revisions:
 *    @li 01-15-2008 JRR Original (somewhat useful) file
 *    @li 10-11-2012 JRR Less original, more useful file with FreeRTOS mutex added
 *    @li 10-12-2012 JRR There was a bug in the mutex code, and it has been fixed
 *
 *  License:
 *    This file is copyright 2012 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//=====================================================================================

// This define prevents this .H file from being included multiple times in a .CPP file
#ifndef _AVR_ADC_H_
#define _AVR_ADC_H_

#include "emstream.h"                       // Header for serial ports and devices
#include "FreeRTOS.h"                       // Header for the FreeRTOS RTOS
#include "task.h"                           // Header for FreeRTOS task functions
#include "queue.h"                          // Header for FreeRTOS queues
#include "semphr.h"                         // Header for FreeRTOS semaphores


//-------------------------------------------------------------------------------------
/** @brief   This class will run the A/D converter on an AVR processor.
 *  @details The class contains a protected pointer  to the serial port for outputs.
 *           Public functions read_once and read_oversampled perform A/D conversions
 *           and emstream prints the A/D conversion value for each channel and the
 *           ADSCRA, ADMUX registers.
 */

class adc
{
    protected:
        /// The ADC class uses this pointer to the serial port to say hello
        emstream* ptr_to_serial;

    public:
        // The constructor sets up the A/D converter for use. The "= NULL" part is a
        // default parameter, meaning that if that parameter isn't given on the line
        // where this constructor is called, the compiler will just fill in "NULL".
        // In this case that has the effect of turning off diagnostic printouts
        adc (emstream* = NULL);

        // This function reads one channel once, returning the result as an unsigned
        // integer; it should be called from within a normal task, not an ISR
        uint16_t read_once (uint8_t);

        // This function reads the A/D lots of times and returns the average. Doing so
```

```
        // implements a crude sort of low-pass filtering that can help reduce noise
        uint16_t read_oversampled (uint8_t, uint8_t);

}; // end of class adc


// This operator prints the A/D converter (see file adc.cpp for details). It's not
// a part of class adc, but it operates on objects of class adc
emstream& operator << (emstream&, adc&);

#endif // _AVR_ADC_H_
```

```
//*****************************************************************************
/** @file adc.cpp
 *    This file contains a simple A/D converter driver.
 *
 *  Revisions:
 *    @li 01-15-2008 JRR Original (somewhat useful) file
 *    @li 10-11-2012 JRR Less original, more useful file with FreeRTOS mutex added
 *    @li 10-12-2012 JRR There was a bug in the mutex code, and it has been fixed
 *
 *  License:
 *    This file is copyright 2015 by JR Ridgely and released under the Lesser GNU
 *    Public License, version 2. It intended for educational use only, but its use
 *    is not limited thereto. */
/*    THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 *    AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 *    IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 *    ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 *    LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
 *    TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 *    OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
 *    CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 *    OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 *    OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. */
//*****************************************************************************

#include <stdlib.h>                         // Include standard library header files
#include <avr/io.h>

#include "rs232int.h"                       // Include header for serial port class
#include "adc.h"                            // Include header for the A/D class


//---------------------------------------------------------------------------------
/** \brief This constructor sets up an A/D converter.
 *  \details The A/D is made ready so that when a method such as @c read_once() is
 *  called, correct A/D conversions can be performed. The ADC Control and Status
 *  Register is set such that the reference voltage source is from AVCC with external
 *  Capacitor at AREF pin and the clock prescaler is at a division factor of 32.
 *  @param p_serial_port A pointer to the serial port which writes debugging info.
 */

adc::adc (emstream* p_serial_port)
{
    ptr_to_serial = p_serial_port;

    ADCSRA |= (1<<ADEN)|(1<<ADPS0)|(1<<ADPS2);
    ADMUX  |= (1<<REFS0);

    // Print a handy debugging message
    DBG (ptr_to_serial, "A/D constructor OK" << endl);
}


//---------------------------------------------------------------------------------
/** @brief   This method takes one A/D reading from the given channel and returns it.
 *  @details This function performs a single A/D conversion based on the paramenter ch. The
 *           selected channel input is read and the ADMUX and ADCSRA registers are set
 *           accordingly. The A/D conversion is initiated and the code waits until
 *           the ADSC bit returns to 0, signifying the conversion is complete. Finally,
 *           the A/D conversion result is stored and the value returned.
 *  @param   ch The A/D channel which is being read must be from 0 to 7
 *  @return  The result of the A/D conversion
 */

uint16_t adc::read_once (uint8_t ch)
```

```
{
    uint16_t result;

    // Set channels depending on input parameter ch
    switch (ch)
    {
      case 0:
        ADMUX  &= ~(1<<MUX4)& ~(1<<MUX3)& ~(1<<MUX2)& ~(1<<MUX1)& ~(1<<MUX0);
        ADCSRA |= (1<<ADSC);
        break;

      case 1:
        ADMUX  &= ~(1<<MUX4)& ~(1<<MUX3)& ~(1<<MUX2)& ~(1<<MUX1);
        ADMUX  |= (1<<MUX0);
        ADCSRA |= (1<<ADSC);
        break;

      case 2:
        ADMUX  &= ~(1<<MUX4)& ~(1<<MUX3)& ~(1<<MUX2)& ~(1<<MUX0);
        ADMUX  |= (1<<MUX1);
        ADCSRA |= (1<<ADSC);
        break;

      case 3:
        ADMUX  &= ~(1<<MUX4)& ~(1<<MUX3)& ~(1<<MUX2);
        ADMUX  |= (1<<MUX1)|(1<<MUX0);
        ADCSRA |= (1<<ADSC);
        break;

      case 4:
        ADMUX  &= ~(1<<MUX4)& ~(1<<MUX3)& ~(1<<MUX1)& ~(1<<MUX0);
        ADMUX  |= (1<<MUX2);
        ADCSRA |= (1<<ADSC);
        break;

      case 5:
        ADMUX  &= ~(1<<MUX4)& ~(1<<MUX3)& ~(1<<MUX1);
        ADMUX  |= (1<<MUX2)|(1<<MUX0);
        ADCSRA |= (1<<ADSC);
        break;

      case 6:
        ADMUX  &= ~(1<<MUX4)& ~(1<<MUX3)& ~(1<<MUX0);
        ADMUX  |= (1<<MUX2)|(1<<MUX1);
        ADCSRA |= (1<<ADSC);
        break;

      case 7:
        ADMUX  &= ~(1<<MUX4)& ~(1<<MUX3);
        ADMUX  |= (1<<MUX2)|(1<<MUX1)|(1<<MUX0);
        ADCSRA |= (1<<ADSC);
        break;

      default :
        break;
    }

    while(ADCSRA & (1<<ADSC))
    {}

    result = (ADCL | (ADCH<<8));
    return (result);
}
```

```
//-------------------------------------------------------------------------------------
/** @brief   This method averages multiple readings from the A/D conversion and returns the average.
 *  \details This function takes a set number of samples from the read_once function and averages them
 *           together to reduce the effects of noise.
 *  @param   channel The A/D channel which is being read must be from 0 to 7.
 *  @param   samples Number of samples to be averaged.
 *  @return  The result of the averaged A/D conversion
 */

uint16_t adc::read_oversampled (uint8_t channel, uint8_t samples)
{
    uint8_t  count;
    uint16_t sum;
    uint16_t average;
    uint16_t result;

    average = 0;
    count = 0;
    sum = 0;

    if (samples > 10)
    {
      samples = 10;
    }

    while (count <= samples)
    {
    result = adc::read_once(channel);
    sum = (result + sum);
    count ++;
    }

    average = (sum/count);
    count = 0;

    return (average);
}

//-------------------------------------------------------------------------------------
/** \brief   This overloaded operator "prints the A/D converter."
 *  \details This prints the A/D control registers ADCSRA and ADMUX as well as the
 *           conversion value from an oversampled conversion with oversampling of 5
 *  @param   serpt Reference to a serial port to which the printout will be printed
 *  @param   a2d   Reference to the A/D driver which is being printed
 *  @return  A reference to the same serial device on which we write information.
 *           This is used to string together things to write with @c << operators
 */

emstream& operator << (emstream& serpt, adc& a2d)
{
    // Print contents of A/D control registers (ADMUX, ADCSRA) in binary
    // Show the current reading of channel 0 with oversampling of 5

//   serpt << PMS ("ADCSRA: ") <<bin <<ADCSRA <<endl;
//   serpt << PMS ("ADMUX: ") <<bin <<ADMUX <<endl;
//   serpt << PMS ("A/D Conversion Value: ") <<dec << a2d.read_oversampled (0,5) <<endl;

    return (serpt);
}
```