MARITIME COLLEGE
STATE UNIVERSITY OF NEW YORK

ENGR 601: Retractable Reefer Cord

Remote Control Application

Prepared for

Dr. Weili Cui

By

Tina Kang

Table of contents

Introduction

Reefer cord reel can improve efficiencies of vessels operations, efficiencies in yard operations, cord life, potential injuries. It can also result in significantly less cost during the new design of vessels and terminals.

For the Retractable Reefer Cord Reel project, Arduino Remote Control App on Android can connect to users wirelessly using Bluetooth, WiFi, or over the web. The aim of this project is to create a simple user interface and easy set up. Attached files include:

- **sim.imo**: Contains all global variables and objects.
- **button.hpp**: Encapsulates a button.
- **button.cpp**: Implementation of the button class in button.hpp file.
- **comparators.hpp**: Functions for comparing values.
- **controllerLimitSwitch.hpp**: Encapsulates a limit switch.
- **controllerLimitSwithch.cpp**: Implementation of the ControllerLimitSwitch class in controllerLimitSwitch.hpp.
- **controllerMotor.cpp**: Implementation of the ControllerMotor class in controller.hpp.
- **controllerMotor.hpp**: Encapsulates a motor.

Method

The remote control app development opens a new door for inexpensive, fast, and easy controller prototyping for the maritime industry.

Arduino analog output (PWM) is used to control the speed of the motor by sending a number between -255 and 255 from the Serial Monitor [2].

```
  const static short MAX_SPEED = 255;
  const static short MIN_SPEED = -255;
```

Pulse Width Modulation is used to provide an efficient and simple method for controlling the speed of DC motor. For the Retractable Reefer Crod, roller motor drives the large diameter cylinder to retract reefer electrical cord. The cylinder is connected by gear to the tensioner pulley rod. As the rod rotates, a threaded bracket traverses back and forth, ensuring the electrical cord stack neatly on the cylinder. Pulse Width Modulation method will change the duration of a pulse with respect to the analog input. The digital circuit is interfaced to microcontroller, varying the speed and power the electric DC motors.

Conclusion

Sim.ino file contains all global variables and objects into the main .INO file. It contains all header files such as controllerLimitSwitch.hpp, controllerMotor.hpp and button.hpp.

When speed changes, sensor output changes in sync with the roller and guide motors. From the output changes Arduino detects change of speed and tries to minimize it by increasing the duty cycle.

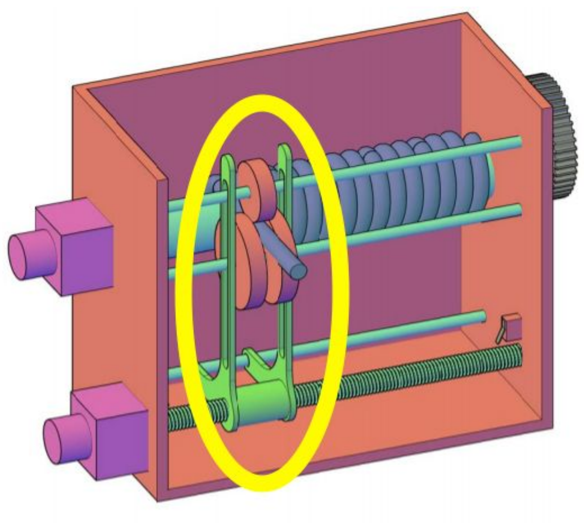


Fig. 1 Roller and guide motors in sync

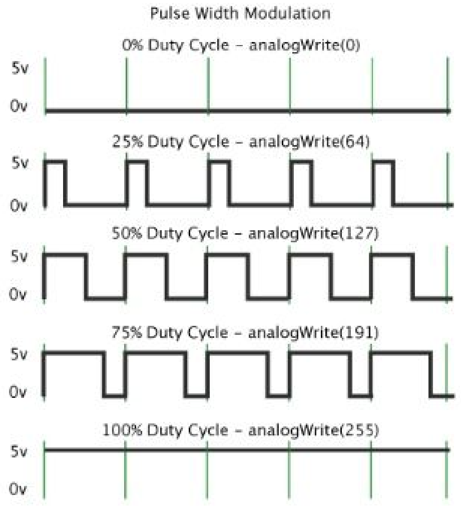

Pulse Width Modulation
0% Duty Cycle - analogWrite(0)
5v
0v
25% Duty Cycle - analogWrite(64)
5v
0v
50% Duty Cycle - analogWrite(127)
5v
0v
75% Duty Cycle - analogWrite(191)
5v
0v
100% Duty Cycle - analogWrite(255)
5v
0v

Fig. 2 Arduino - Pulse Width Modulation [1]

For the controller motor, Remote Control Program allows user to instantiate, change direction, stop, change speed of the roller and guide motors. The main advantage in using a DC motor is that the Speed-Torque relationship can be varied to almost any useful form. To achieve the speed control an electronic technique called Pulse Width Modulation is used which generates High and Low pulses. These pulses vary the speed in the motor.

The electric and electromechanical methods are less adaptive so Pulse Width Modulation is used for speed control.

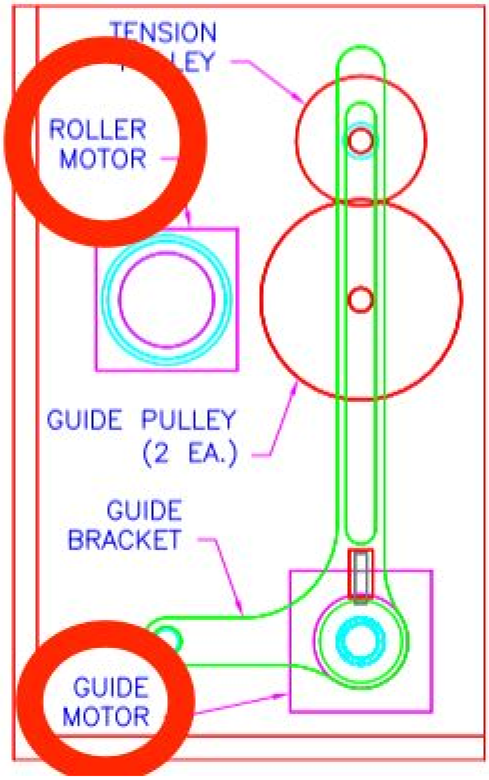


Fig. 4 Controller Motor

Additionally, two limit switches on either end of the threaded rod were used to program reversing DC motor [3]. When the guided bracket approaches the end of the encasement, the limit switch engages, revering the guide motor. For object avoidance, the ultrasonic or the ir range finders do not always see and objects.

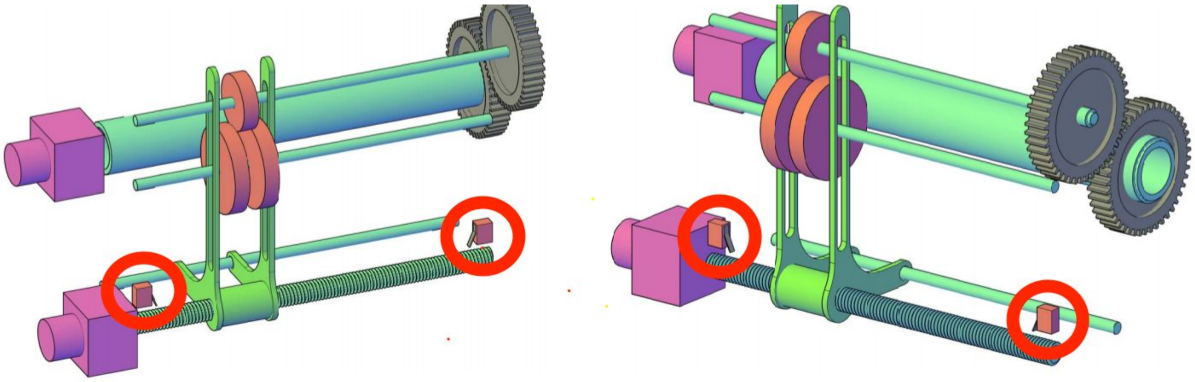


Fig. 5 Controller Limit Switch

As for the buttons, remote control application has five functions [4]. Up, down, left, right, and stop. H-Bridge will be used as an electronic circuit that enables a voltage to be applied across a load in either direction. These circuits are often used in Robotics and other applications to allow DC motors to move [5].

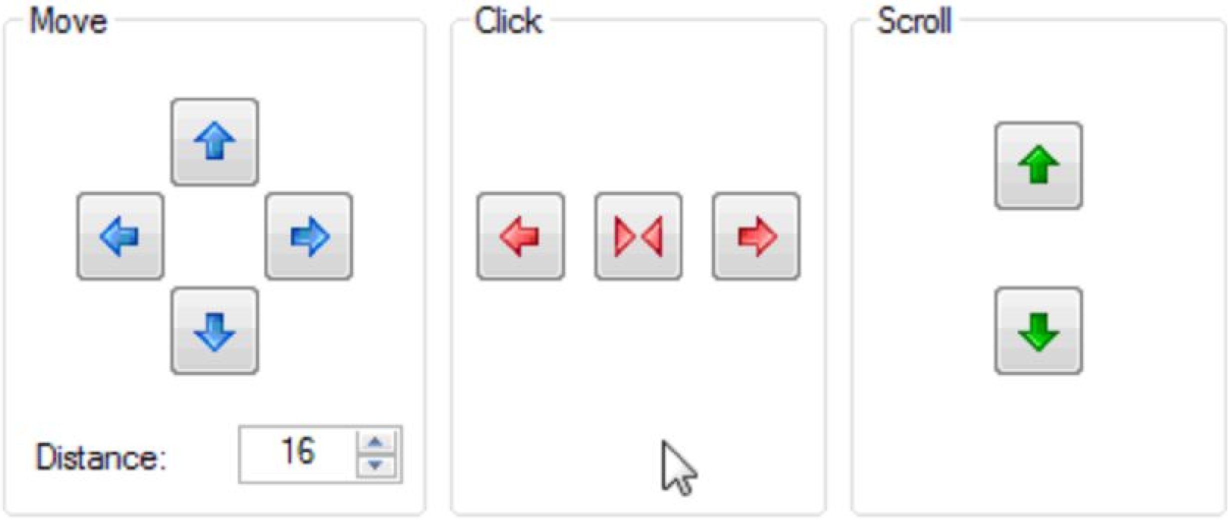
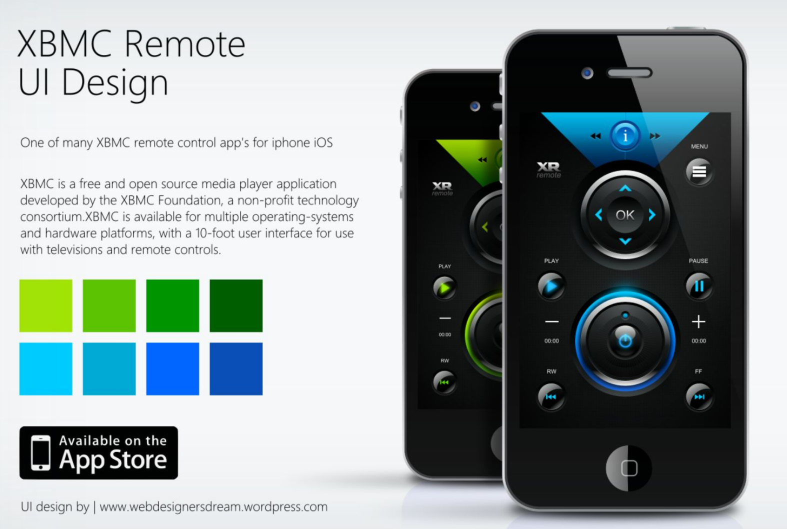


Fig. 6 Buttons

In addition, a comparator is used to compare a measurable quantity with a reference or standard such as two voltages or currents [6]. It outputs a digital signal showing the results. Contains functions for comparing values.

More friendly user experience design and Amazon fire stick remote app model are suggested for future modification. User Experience Design (UX) enhances user satisfaction by improving the usability, accessibility, and pleasure provided in the interaction with the product.



XBMC Remote UI Design

One of many XBMC remote control app's for iphone iOS

XBMC is a free and open source media player application developed by the XBMC Foundation, a non-profit technology consortium.XBMC is available for multiple operating-systems and hardware platforms, with a 10-foot user interface for use with televisions and remote controls.

UI design by | www.webdesignersdream.wordpress.com

Fig. 7 User Experience Design

Also, Fire TV stick model for the remote control app is strongly encouraged due to its simplicity in both designing and programming.

Fig. 8 Fire TV stick

References

1. **Arduino:** https://www.arduino.cc/
2. **Arduino PWM**: https://www.arduino.cc/en/Tutorial/PWM
3. **DC Motor:** https://www.tutorialspoint.com/arduino/arduino_dc_motor.htm
4. **Arduino - Button:** https://www.arduino.cc/en/Tutorial/Button
5. **DC Motor Control Using an H-Bridge:** https://itp.nyu.edu/physcomp/labs/motors-and-transistors/dc-motor-control-using-an-h-bridge/
6. **Comparator circuits:** http://www.bristolwatch.com/ele2/comparator.htm

Appendix

- **sim.imo**: Contains all global variables and objects.

```
#include "controllerLimitSwitch.hpp"
#include "controllerMotor.hpp"
#include "button.hpp"

const short POS_SPEED_RATE = 10;
const short NEG_SPEED_RATE = -10;

//controllers
ControlledMotor horizontal;
ControlledMotor vertical;
ControlledLimitSwitch leftLimit;
ControlledLimitSwitch rightLimit;
//buttons
Button up;
Button dn;
Button lt;
Button rt;
Button stp;

void setup() {
    //set up the controller classes with their respective pins
    horizontal = ControlledMotor(1);
    vertical = ControlledMotor(2);
    leftLimit = ControlledLimitSwitch(3);
    rightLimit = ControlledLimitSwitch(4);
    up = Button(5);
    dn = Button(6);
    lt = Button(7);
    rt = Button(8);
    stp = Button(9);
}

inline void moveRight() {
    //if not already moving right move to right
    if (horizontal.stopped() || horizontal.getSpeed() <= NEG_SPEED_RATE) {
        horizontal.changeSpeed(POS_SPEED_RATE);
    }
}
```

```
inline void moveLeft() {
   //if not already moving left move left
   if (horizontal.stopped() || horizontal.getSpeed() >= POS_SPEED_RATE) {
      horizontal.changeSpeed(NEG_SPEED_RATE);
   }
}

inline void moveUp() {
   //if not already moving up move up
   if (vertical.stopped() || vertical.getSpeed() <= NEG_SPEED_RATE) {
      vertical.changeSpeed(POS_SPEED_RATE);
   }
}

inline void moveDown() {
   //if not already moving down move down
   if (vertical.stopped() || vertical.getSpeed() >= POS_SPEED_RATE) {
      vertical.changeSpeed(NEG_SPEED_RATE);
   }
}

inline void stop() {
   //stop all motors
   horizontal.stop();
   vertical.stop();
}

void loop() {
   //determine move
   if (up.pressed()) moveUp();
   else if (dn.pressed()) moveDown();
   else if (lt.pressed()) moveLeft();
   else if (rt.pressed()) moveRight();
   else if (stp.pressed()) stop();

   //check if either limit is hit and change direction
   if (!horizontal.stopped() && (leftLimit.hitLimit() || rightLimit.hitLimit())) {
      horizontal.changeDirection();
   }
}
```

- **button.hpp**: Encapsulates a button.

```
/*
  button.hpp
  encapsulates a button
*/

#ifndef BUTTON_HPP
#define BUTTON_HPP

#include "Arduino.h"
```

```
/*
  Button
  keeps track of button
*/
class Button {
  short _pin;
public:
  Button();
  Button(const short pin);
  const bool pressed();
};

#endif
```

- **button.cpp**: Implementation of the button class in button.hpp file.

```
/*
  button.cpp
  implementation of the button class in button.hpp
*/

#include "button.hpp"

/*
  constructor
  default doesn't do anything
*/
Button::Button() {}

/*
  constructor
  instantiates the button
*/
Button::Button(const short pin) {
  _pin = pin;
  pinMode(pin, INPUT);
}

/*
  pressedt
  check if the button has been pressed
*/
const bool Button::pressed() {
  return digitalRead(_pin) == HIGH;
}
```

- **comparators.hpp**: Functions for comparing values.

```
/*
  comparators.hpp
  functions for comparing values
*/
```

```
#ifndef COMPARATORS_HPP
#define COMPARATORS_HPP

template <typename T>
__attribute__ ((pure)) inline T min (T a, T b) {
   return a < b ? a : b;
}

template <typename T>
__attribute__ ((pure)) inline T max (T a, T b) {
   return a > b ? a : b;
}

#endif
```

- **controllerLimitSwitch.hpp**: Encapsulates a limit switch.

```
/*
  controllerLimitSwitch.hpp
  encapsulates a limit switch
*/

#ifndef CONTROLLEDLIMITSWITCH_HPP
#define CONTROLLEDLIMITSWITCH_HPP

#include "Arduino.h"

/*
  ControlledLimitSwitch
  keeps track of limit switch
*/
class ControlledLimitSwitch {
  const static short LIMIT = 1;
  const static short NOT_LIMIT = 0;
  short _pin;
public:
  ControlledLimitSwitch();
  ControlledLimitSwitch(const short pin);
  const bool hitLimit();
};
```

- **controllerLimitSwithch.cpp**: Implementation of the ControllerLimitSwitch class in controllerLimitSwitch.hpp.

```
/*

  ControllerLimitSwitch.cpp
  implementation of the ControlledLimitSwitch class in controllerLimitSwitch.hpp
*/

#include "controllerLimitSwitch.hpp"
```

```
/*
  constructor
  default doesn't do anything
*/
ControlledLimitSwitch::ControlledLimitSwitch() {}

/*
  constructor
  instantiates the limit switch
*/
ControlledLimitSwitch::ControlledLimitSwitch(const short pin) {
  _pin = pin;
  pinMode(pin, INPUT);
}

/*
  hitLimit
  check if the limit has been hit
*/
const bool ControlledLimitSwitch::hitLimit() {
  return digitalRead(_pin) == LIMIT ? true : false;
}
```

- **controllerMotor.cpp**: Implementation of the ControllerMotor class in controller.hpp.

```
/*
  ControllerMotor.cpp
  implementation of the ControlledMotor class in controller.hpp
*/

#include "controllerMotor.hpp"

/*
  constructor
  default doesn't do anything
*/
ControlledMotor::ControlledMotor() {}


/*
  constructor
  instantiates the motor
*/
ControlledMotor::ControlledMotor(const short pin, const short speed = NO_SPEED) {
  _pin = pin;
  _speed = speed;
  pinMode(pin, OUTPUT);
  digitalWrite(pin, speed);
}

/*
```

```
  changeDirection
  changes direction of motor
*/
void ControlledMotor::changeDirection() {
   changeSpeed(0 - _speed);
}

/*
  stop
  stops the motor
*/
void ControlledMotor::stop() {
  digitalWrite(_pin, NO_SPEED);
}

/*
  stopped
  check if motor is stopped
*/
const bool ControlledMotor::stopped() {
  return _speed == NO_SPEED;
}

/*
  changeSpeed
  changes the speed of the motor
*/
const short ControlledMotor::changeSpeed(const short speed) {
  _speed = max(MIN_SPEED, min(MAX_SPEED, speed));
  digitalWrite(_pin, _speed);

  return _speed;
}

/*
  getSpeed
  returns the current speed of the dc motor
*/
const short ControlledMotor::getSpeed(){
  return _speed;
}
```

- **controllerMotor.hpp**: Encapsulates a motor.

```
/*
  controllerMotor.hpp
  encapsulates a motor
*/

#ifndef CONTROLLEDMOTOR_HPP
#define CONTROLLEDMOTOR_HPP

#include "Arduino.h"
```

```
/*
  ControlledMotor
  keeps track of and manipulates a motor
*/
class ControlledMotor {
  const static short MAX_SPEED = 255;
  const static short MIN_SPEED = -255;
  const static short NO_SPEED = 0;
  short _speed;
  short _pin;
public:
  ControlledMotor();
  ControlledMotor(const short pin, const short speed = NO_SPEED);
  void changeDirection();
  void stop();
  const short changeSpeed(const short speed);
  const short getSpeed();
  const bool stopped();
};

#endif
```