# 1 Control

## 1.1 Main File

```
#include "PWM.h"
#include "PID.h"
#include "utils.h"

PID yawPID(1,0,0);
PID pitchPID(1,0,0);
PID rollPID(1,0,0);

//controller values
float throttle;
float targetYaw;
float targetPitch;
float targetRoll;
//gyro values
float gyroYaw;
float gyroPitch;
float gyroRoll;
//PID output values
float pidYaw;
float pidPitch;
float pidRoll;

void setup() {
  // put your setup code here, to run once:
  init_pwm();
}

void loop() {
  // put your main code here, to run repeatedly:
  //if(mpuInterupt)
  //{

    //get gyro data
    gyroYaw = 0;
    gyroPitch = 0;
    gyroRoll = 0;

    //get throttle data
    throttle = 0;
    targetYaw = 0;
    targetPitch = 0;
    targetRoll = 0;

    //apply PID
    pidYaw = yawPID.updatePID(targetYaw, gyroYaw, DELTA_TIME);
    pidPitch = pitchPID.updatePID(targetPitch, gyroPitch, DELTA_TIME);
    pidRoll = rollPID.updatePID(targetRoll, gyroRoll, DELTA_TIME);
    //update motors
    setMotors (throttle, pidYaw, pidPitch, pidRoll);
  //}


  //send telemetry
}
```

## 1.2 PWM output Funtions

```c
#include "Definitions.h"
#include "PWM.h"
#include "Arduino.h"
void init_pwm(void)
{
    /* TIMER 1 */
    DDRB |= _BV(PB5); /* PWM 1A out (pin 9 on pro micro)*/
    DDRB |= _BV(PB6); /* PWM 1B out (pin 10 on pro micro)*/
    DDRB |= _BV(PB7); /* PWM 1C out (non existant on pro micro )*/
    TCCR1A =    _BV(WGM11) | /* fast PWM/MAX */
                _BV(WGM12) | /* fast PWM/MAX */
                _BV(WGM13) | /* fast PWM/MAX */
                _BV(COM1A1)| /* A output enabled*/
                _BV(COM1B1)| /* C output enabled*/
                _BV(COM1C1); /* B output enabled*/
    TCCR0B =  _BV(CS11)    ; /*/8 prescaling */
    ICR1 = TIMER_TOP;
    /* TIMER 3 */
    DDRC |= _BV(PC6); /* PWM 3A out (pin 5 on pro micro)*/
    TCCR3A =  _BV(WGM31) | /* fast PWM/MAX */
              _BV(WGM32) | /* fast PWM/MAX */
              _BV(WGM33) |
              _BV(COM3A1); /* A output enabled*/
    TCCR3B =  _BV(CS31)   ;
    ICR3 = TIMER_TOP;
    pwm_duty(LEFT_FRONT_MOTOR, MIN_MOTOR_SPEED);
    pwm_duty(RIGHT_FRONT_MOTOR, MIN_MOTOR_SPEED);
    pwm_duty(LEFT_REAR_MOTOR, MIN_MOTOR_SPEED);
    pwm_duty(RIGHT_REAR_MOTOR, MIN_MOTOR_SPEED);
}

void pwm_duty(uint8_t motor,uint16_t duty)
{   //duty is currently in ms, we need to convert it to a value in the correct range.
    //range ms: 1000-2000 , range registers: 2000-4000 therefore multiply by 2
    duty = duty*2;
    if(duty>PWM_DUTY_MAX) duty = PWM_DUTY_MAX;
    else if(duty<PWM_DUTY_MIN) duty = PWM_DUTY_MIN;

    switch(motor)
    {
      case LEFT_FRONT_MOTOR:
        OCR1A = duty;
      case RIGHT_FRONT_MOTOR:
        OCR1B = duty;
      case LEFT_REAR_MOTOR:
        OCR1C = duty;
      case RIGHT_REAR_MOTOR:
        OCR3A = duty;
    }
}

void setMotors (float throttle, float yaw, float pitch, float roll)
{
  //reasons for these particular equations are given below
  pwm_duty(LEFT_FRONT_MOTOR,  (uint16_t)(throttle - roll - pitch + yaw));
  pwm_duty(RIGHT_FRONT_MOTOR, (uint16_t)(throttle + roll - pitch - yaw));
  pwm_duty(LEFT_REAR_MOTOR,   (uint16_t)(throttle - roll + pitch - yaw));
  pwm_duty(RIGHT_REAR_MOTOR,  (uint16_t)(throttle + roll + pitch + yaw));
}
/*
CW motors     A,C
```

```
CCW motors    D,B

           Front
           +1 pitch
          C   D
-1 roll    \-/     +1 roll    right
           /-\
          B   A
           -1 pitch

c = throttle - roll + pitch + yaw
d = throttle + roll + pitch - yaw
b = throttle - roll - pitch - yaw
a = throttle + roll - pitch + yaw
[1]http://robotics.stackexchange.com/questions/5116/
how-to-find-a-solution-for-quadcopter-pid-control
//seems to be inverted pitch so we changed it
  pwm_duty(LEFT_FRONT_MOTOR, (uint16_t)(throttle - roll + pitch + yaw));
  pwm_duty(RIGHT_FRONT_MOTOR, (uint16_t)(throttle + roll + pitch - yaw));
  pwm_duty(LEFT_REAR_MOTOR, (uint16_t)(throttle - roll - pitch - yaw));
  pwm_duty(RIGHT_REAR_MOTOR, (uint16_t)(throttle + roll - pitch + yaw));
*/
```

## 1.3 PID Class (Found online)

```
/*
  PID.cpp - Library for implementing a PID control loop. Used to ensure engines don't overshoot when reaching target roll/pitch.
  Created by Myles Grant <myles@mylesgrant.com>
  Based on:
  See also: https://github.com/grantmd/QuadCopter

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program. If not, see <http://www.gnu.org/licenses/>.
*/

//#include "WProgram.h"
#include "PID.h"

PID::PID(){
  iState = 0;
  last = 0;

  pgain = 0;
  igain = 0;
  dgain = 0;
}

PID::PID(float p, float i, float d){
  PID();

  pgain = p;
  igain = i;
  dgain = d;
}

// get the P gain
float PID::getP(){
  return pgain;
}

// get the I gain
float PID::getI(){
  return igain;
}

// get the D gain
float PID::getD(){
  return dgain;
}

// set the P gain and store it to eeprom
void PID::setP(float p){
  pgain = p;
  //writeFloat(p, pgainAddress);
}

// set the I gain and store it to eeprom
void PID::setI(float i){
  igain = i;
  //writeFloat(i, igainAddress);
}

// set the D gain and store it to eeprom
void PID::setD(float d){
  dgain = d;
  //writeFloat(d, dgainAddress);
}

float PID::updatePID(float target, float cur, float deltaTime){
  // these local variables can be factored out if memory is an issue,
  // but they make it more readable
  float error;
  float windupGuard;

  // determine how badly we are doing
  error = target - cur;

  // the pTerm is the view from now, the pgain judges
  // how much we care about error at this instant.
  pTerm = pgain * error;

  // iState keeps changing over time; it's
  // overall "performance" over time, or accumulated error
  iState += error * deltaTime;

  // to prevent the iTerm getting huge despite lots of
  //  error, we use a "windup guard"
  // (this happens when the machine is first turned on and
  // it cant help be cold despite its best efforts)

  // not necessary, but this makes windup guard values
  // relative to the current iGain
  windupGuard = WINDUP_GUARD_GAIN / igain;

  if (iState > windupGuard)
    iState = windupGuard;
  else if (iState < -windupGuard)
    iState = -windupGuard;
  iTerm = igain * iState;

  // the dTerm, the difference between the temperature now
  //  and our last reading, indicated the "speed,"
  // how quickly the temp is changing. (aka. Differential)
  dTerm = (dgain * (cur - last)) / deltaTime;

  // now that we've use lastTemp, put the current temp in
  // our pocket until for the next round
  last = cur;

  // the magic feedback bit
  return pTerm + iTerm - dTerm; //why is this a minus ?
}

void PID::resetError(){
  iState = 0;
}
//[1]http://robot-kingdom.com/pid-controller-tutorial-for-robots/
//[2]https://github.com/grantmd/QuadCopter
```

## 1.4 Other utilities for PID code

```
#include "utils.h"
#include "Definitions.h"
float rawToAngle(int controlIn)
{
 float output = controlIn - 512;
 output = output/N_ANGLE;
 return output;
}

float rawToThrottle(int controlIn)
{
 float output = controlIn - 512;
 if(output > 0)
 {
  output = (output/N_THROTTLE)+1000;
  return output;
 }
 return 0;
}
```

# 2 Sensing

## 2.1 Gyroscope (example code we will use for getting angles from gyro)

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using
    DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/
    i2cdevlib
//
// Changelog:
//      2013-05-08 - added seamless Fastwire support
//                 - added note about gyro calibration
//      2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility error
//      2012-06-20 - improved FIFO overflow handling and simplified read process
//      2012-06-19 - completely rearranged DMP initialization code and simplification
//      2012-06-13 - pull gyro and accel data from FIFO packet instead of reading directly
//      2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING
//      2012-06-05 - add gravity-compensated initial reference frame acceleration output
//                 - add 3D math helper file to DMP6 example sketch
//                 - add Euler output and Yaw/Pitch/Roll output formats
//      2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)
//      2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250
//      2012-05-30 - basic DMP initialization working

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

```

```
/* ============================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
 * ============================================ */

/* ============================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
   when using Serial.write(buf, len). The Teapot output uses this method.
   The solution requires a modification to the Arduino USBAPI.h file, which
   is fortunately simple, but annoying. This will be fixed in the next IDE
   release. For more info, see these links:

   http://arduino.cc/forum/index.php/topic,109987.0.html
   http://code.google.com/p/arduino/issues/detail?id=958
 * ============================================ */



// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
//#define OUTPUT_READABLE_QUATERNION

// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
// (in degrees) calculated from the quaternions coming from the FIFO.
// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal_lock)
//#define OUTPUT_READABLE_EULER

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
//#define OUTPUT_TEAPOT



#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
```

```
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };



// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================


volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
    mpuInterrupt = true;
}



// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================

void setup() {
    // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation difficulties
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
    #endif

    // initialize serial communication
    // (115200 chosen because it is required for Teapot Demo output, but it's
    // really up to you depending on your project)
    Serial.begin(115200);
    while (!Serial); // wait for Leonardo enumeration, others continue immediately

    // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio
    // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
    // the baud timing being too misaligned with processor ticks. You must use
    // 38400 or slower in these cases, or use some kind of external separate
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();

    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and demo: "));
    while (Serial.available() && Serial.read()); // empty buffer
    while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(220);
    mpu.setYGyroOffset(76);
    mpu.setZGyroOffset(-85);
    mpu.setZAccelOffset(1788); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));


  // set up ISR         cause          ISR_name     mode
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;


        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }

    // configure LED for output
    pinMode(LED_PIN, OUTPUT);
}



// ================================================================
// ===                    MAIN PROGRAM LOOP                     ===
// ================================================================

void loop() {

    // if programming failed, don't try to do anything
    if (!dmpReady) return;



    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {
        // other program behavior stuff here
        // .
        // .
        // .
        // if you are really paranoid you can frequently test in between other
        // stuff to see if mpuInterrupt is true, and if so, "break;" from the
        // while() loop to immediately process the MPU data
        // .
        // .
        // .
    }

    // reset interrupt flag and get INT_STATUS byte
    mpuInterrupt = false;
    mpuIntStatus = mpu.getIntStatus();


    // get current FIFO count
    fifoCount = mpu.getFIFOCount();

    // check for overflow (this should never happen unless our code is too inefficient)
    if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        Serial.println(F("FIFO overflow!"));


    // otherwise, check for DMP data ready interrupt (this should happen frequently)
    } else if (mpuIntStatus & 0x02) {
        // wait for correct available data length, should be a VERY short wait
        while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

        // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // track FIFO count here in case there is > 1 packet available
        // (this lets us immediately read more without waiting for an interrupt)
        fifoCount -= packetSize;

        #ifdef OUTPUT_READABLE_QUATERNION
            // display quaternion values in easy matrix form: w x y z
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            Serial.print("quat\t");
            Serial.print(q.w);
            Serial.print("\t");
            Serial.print(q.x);
            Serial.print("\t");
            Serial.print(q.y);
            Serial.print("\t");
            Serial.println(q.z);
        #endif

        #ifdef OUTPUT_READABLE_EULER
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetEuler(euler, &q);
            Serial.print("euler\t");
            Serial.print(euler[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(euler[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(euler[2] * 180/M_PI);
        #endif

        #ifdef OUTPUT_READABLE_YAWPITCHROLL
            // display Euler angles in degrees
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
        #endif

        #ifdef OUTPUT_READABLE_REALACCEL
            // display real acceleration, adjusted to remove gravity
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            Serial.print("areal\t");
            Serial.print(aaReal.x);
            Serial.print("\t");
            Serial.print(aaReal.y);
            Serial.print("\t");
            Serial.println(aaReal.z);
        #endif

        #ifdef OUTPUT_READABLE_WORLDACCEL
            // display initial world-frame acceleration, adjusted to remove gravity
            // and rotated based on known orientation from quaternion
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            mpu.dmpGetAccel(&aa, fifoBuffer);
            mpu.dmpGetGravity(&gravity, &q);
            mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
            Serial.print("aworld\t");
            Serial.print(aaWorld.x);
            Serial.print("\t");
            Serial.print(aaWorld.y);
            Serial.print("\t");
            Serial.println(aaWorld.z);
        #endif

        #ifdef OUTPUT_TEAPOT
            // display quaternion values in InvenSense Teapot demo format:
            teapotPacket[2] = fifoBuffer[0];
            teapotPacket[3] = fifoBuffer[1];
            teapotPacket[4] = fifoBuffer[4];
            teapotPacket[5] = fifoBuffer[5];
            teapotPacket[6] = fifoBuffer[8];
            teapotPacket[7] = fifoBuffer[9];
            teapotPacket[8] = fifoBuffer[12];
            teapotPacket[9] = fifoBuffer[13];
            Serial.write(teapotPacket, 14);
            teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```

## 2.2 IR sensor

```
#include <avr/io.h>
#include <util/delay.h>
#include <math.h>
#include "debug.h"

// avr-gcc -mmcu=atmega644p -DF_CPU=12000000 -Wall -Os -Wl,-u,vfprintf -lprintf_flt -lm IR.c -o IR.elf
// avr-objcopy -O ihex IR.elf IR.hex
// avrdude -c usbasp -p m644p -U flash:w:IR.hex

void init_adc(void)
{
 ADCSRA |= _BV(ADPS2) | _BV(ADPS1) | _BV(ADEN);
 ADMUX |= _BV(REFS0);
}

uint16_t read_adc(void)
{
 ADCSRA |= _BV(ADSC);
 while(ADCSRA & _BV(ADSC));

 return ADC;
}

double to_distance(uint16_t adc_value)
{
 double distance, volts;
 volts = (adc_value*3.3)/1024;
 distance = 24/volts;
 return distance;
}

int main(void)
{
 uint16_t result;
 double voltage;

 init_debug_uart0();
 init_adc();

 for (;;)
 {
  result = read_adc();

  voltage = to_distance(result);
  printf("%.6f\n",voltage);
  result = 0x0000;
  voltage = 0;

  _delay_ms(1000);
 }
}
```

# 3 Communication

## 3.1 Communications Code from Base station

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <math.h>

#include "rfm12.h"
#include "basestation_comms.h"

uint8_t encryption_key;

int main(void)
{
 // Initialise rfm12 and interrupts
 rfm12_init();
 sei();

 encryption_key = 5;

 // Send test data
 uint16_t testdata = 0;

 while (1)
 {
  rfm12_tick();

  #if UPLINK_TEST
   Send_data(OP_ROLL, testdata);
   testdata++;
   if (testdata == 1024) break;
   _delay_ms(1000);
  #endif

 }
 while (1) {};
}

//!
/*  Process data and send it to the transceiver for transmission.
 If encryption is enabled in the basestation_comms.h then the data will be
    encrypted.
 The 10-bit data is encoded such that the 2 MSBs are stored in the packet type.
*/
void Send_data(uint8_t type, uint16_t data)
{
 // Combine packet type and data into a single 16-bit int
 uint16_t totalpacket;
 totalpacket = type;
 totalpacket = (totalpacket << DATA_BIT_SIZE) + data;

 // Encrypt data
 #if ENCRYPTION_ENABLED
  totalpacket = Encrypt_data(totalpacket);
 #endif

 // Split 16-bit packet into two 8-bit ints - packet type and data
 uint8_t datapacket;
 Encode_data(&type, &datapacket, totalpacket);

 // Send packet to the buffer for transmission
 rfm12_tx(sizeof(datapacket), type, &datapacket);
}

//!
/*  Encode the total packet into the type and data
*/
void Encode_data(uint8_t* type, uint8_t* data, uint16_t totalpacket)
{
 // Data is equal to the 8 LSBs
 *data = totalpacket;

 // Type, encryption key and 2 bits of data are held in the 8 MSBs
 *type = (totalpacket >> DATA_BIT_SIZE);
}

//!
/*  Encrypt the packet type and data using an encryption key.
 This encryption key changes every time the data is encrypted.
*/
uint16_t Encrypt_data(uint16_t packet)
{
 // Retrieve bits that are shifted out when the right shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet & ((uint8_t) pow(2, encryption_key) - 1));

 // Get completely rotated bits by adding the shifted out bits to the
 // original packet right-shifted by the required number of bits.
 uint16_t encrypted_packet;
 encrypted_packet = (packet >> encryption_key) + (rotated_out_bits << (
     COMMAND_BIT_SIZE + DATA_BIT_SIZE - encryption_key));

 // Add on the encryption key to the MSBs of the packet
 encrypted_packet = encrypted_packet + (encryption_key << (COMMAND_BIT_SIZE +
     DATA_BIT_SIZE));

 // Adjust encryption key for next transmission
 encryption_key = (encryption_key < 3) ? encryption_key + 5 : encryption_key - 3;
 if (encryption_key == 0) encryption_key = 5;

 return encrypted_packet;
}
```

### 3.2 Communications code from the drone

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "rfm12.h"
#include "drone_comms.h"

int main(void)
{
 // Initialise rfm12 and interrupts
 rfm12_init();
 sei();

 uint8_t receivedpackettype;
 uint16_t receiveddata;

 while (1)
 {
  rfm12_tick();

  // Wait for data to be fully received
  if (rfm12_rx_status() == STATUS_COMPLETE)
  {
   // Get the received packet type and data
   receivedpackettype = rfm12_rx_type();
   receiveddata = rfm12_rx_buffer();

   // Decrypt (if enabled) and extract 10-bit data and packet type from the received packet
   Retrieve_data(&receivedpackettype, &receiveddata);

   #if UPLINK_TEST
    // Send data to UART
   #endif

  }
 }
}

void Retrieve_data(uint8_t* type, uint16_t* data)
{
 // Combine packet type and data into a single 16-bit int
 uint16_t totalpacket;
 totalpacket = type;
 totalpacket = (totalpacket << DATA_BIT_SIZE) + data;

 #if ENCRYPTION_ENABLED
  // Decrypt the received packet
  totalpacket = Decrypt_data(totalpacket);
 #endif // ENCRYPTION_ENABLED


 // Split the decrypted packet into the data and the packet type
 Decode_data(type, data, totalpacket);
}

uint16_t Decode_data(uint8_t* type, uint16_t* data, uint16_t totalpacket)
{
 // Get 10-bit data from the 16 bit packet
 *data = totalpacket & (uint16_t)1023;

 // Get packet type
 *type = (totalpacket >> DATA_BIT_SIZE);
}

uint16_t Decrypt_data(uint16_t packet)
{
 // Retrieve the encryption key
 uint8_t encryption_key;
 encryption_key = (packet >> (DATA_BIT_SIZE + COMMAND_BIT_SIZE));

 // Retrieve bits that are shifted out when the left shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet >> (DATA_BIT_SIZE + COMMAND_BIT_SIZE - encryption_key));

 // Get completely rotated bits by adding the shifted out bits to the
 // original packet left-shifted by the required number of bits.
 // It is & with a sequence of 1s to remove the encryption key from the overall packet
 uint16_t decrypted_packet;
 decrypted_packet = (((packet << encryption_key) & (pow(2, DATA_BIT_SIZE + COMMAND_BIT_SIZE) - 1)) + rotated_out_bits;

 return decrypted_packet;
}
```

## 3.3 Packet encryption and encoding

```
#include <avr/io.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "../comms.h"

// Mohammed's UART code
#define BAUD 9600                                  // define baud
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)           // set baud rate value for UBRR

void init_uart1()// initialize UART
{
 //1. set the baud rate, lets configure to 9600;
 // set the baud rate registers Ref: [1],[2]
 UBRR0H = BAUDRATE >> 8;// UBRRnH is 8 bits left
 UBRR0L = BAUDRATE;

 //2. setting up data packet: 8 bits ,no parity 1 stop bit
 // setting 8 bits got to UCSCR register Ref:[3], pg 185 of data sheet

 UCSR0C = _BV(UCSZ00) | _BV(UCSZ01); // 8 bits, USBS1 = 0 for 1 stop bit

           // note: havnt set up the stop bit in Ref [2] slides
           // 3. from Ref[2] we now enable Transmission and receive n UCSRnB register
 UCSR0B = _BV(TXEN0) | _BV(RXEN0);
}

void uart_transmit(char data)
{
 while (!(UCSR0A & _BV(UDRE0))); //  data register enable bit is 1 if tx buffer is empty
           // if its 1 we load data onto UDR- Uart Data Register(buffer)
 UDR0 = data;
}

void send_string(char *str)
{
 int i;
 for (i = 0; str[i]; i++) uart_transmit(str[i]);
}//***************void test_encode_decode()

void test_encode()
{
 // Encode
 send_string("Encoding data\n");
 // 998 = 11 1110 0110
 uint16_t ADCoutput;
 ADCoutput = 998;
 // 1110 0110
 uint8_t lsb8;
 lsb8 = ADCoutput;
 // 11 0000 0000
 uint16_t msb2;
 msb2 = ADCoutput - lsb8;
 // 00 0000 0011
 msb2 = msb2 >> 8;
 // 0000 1011
 uint8_t packettype;
 packettype = (2 << 2) + msb2;

 // Sent data: [00001011] [1110 0110]

 // Decode
 send_string("Decoding data\n");
 // 0000 1011 1110 0110
 uint16_t receiveddata;
 receiveddata = (packettype << 8) + lsb8;
 // 0000 0011 1110 0110
 uint16_t decodeddata;
 decodeddata = receiveddata & (uint16_t)1023;
 uint8_t decodedpackettype;
 decodedpackettype = (packettype >> 2);

 char sendData[30];
 sprintf(sendData, "Decoded data: %d\n", decodeddata);
 send_string(sendData);
}

uint16_t test_decrypt(uint16_t packet)
{
 // Retrieve the encryption key
 uint8_t encryption_key;
 encryption_key = (packet >> (DATA_BIT_SIZE + COMMAND_BIT_SIZE));

 char sendData0[30];
 sprintf(sendData0, "Encryption key: %u\r\n", encryption_key);
 send_string(sendData0);

 // Remove the encryption key from the packet
 packet = (packet & ((uint16_t)pow(2, DATA_BIT_SIZE + COMMAND_BIT_SIZE) - 1));

 // Retrieve bits that are shifted out when the left shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet >> (DATA_BIT_SIZE + COMMAND_BIT_SIZE - encryption_key));
```

```c

 // Get completely rotated bits by adding the shifted out bits to the
 // original packet left-shifted by the required number of bits.
 // It is & with a sequence of 1s to remove the encryption key from the overall packet
 uint16_t decrypted_packet;
 decrypted_packet = ((packet << encryption_key) & ((uint16_t)pow(2, DATA_BIT_SIZE + COMMAND_BIT_SIZE) - 1)) + rotated_out_bits;

 char sendData2[30];
 sprintf(sendData2, "Decrypted: %u\r\n", decrypted_packet);
 send_string(sendData2);

 return decrypted_packet;
}

uint16_t test_encrypt(uint16_t packet)
{
 //uint16_t packet;
 //packet = 343 + (6 << DATA_BIT_SIZE);

 char sendData1[50];
 sprintf(sendData1, "\r\nEncrypting %u\r\n", packet);
 send_string(sendData1);

 uint8_t encrypt_key;
 encrypt_key = 4;

 //send_string("Encrypting 868 with a packet of 2 by 2 bits\r\n");

 // Retrieve bits that are shifted out when the right shift is done
 uint8_t rotated_out_bits;
 rotated_out_bits = (packet & ((uint8_t)pow(2, encrypt_key) - 1));

 // Get completely rotated bits by adding the shifted out bits to the original packet right-shifted by the required number of bits.
 uint16_t encrypted_packet = (packet >> encrypt_key) + (rotated_out_bits << (COMMAND_BIT_SIZE + DATA_BIT_SIZE - encrypt_key));

 // Add on the encryption key to the MSBs of the packet
 encrypted_packet = encrypted_packet + (encrypt_key << (COMMAND_BIT_SIZE + DATA_BIT_SIZE));

 char sendData2[30];
 sprintf(sendData2, "Encrypted: %u\r\n", encrypted_packet);
 send_string(sendData2);

 return encrypted_packet;
}

int main(void)
{
 init_uart1();

 uint16_t testpacket, result;
 for (testpacket = 0; testpacket < 8192; testpacket++)
 {
  result = test_encrypt(testpacket);
  if (test_decrypt(result) != testpacket)
  {
   send_string("Error!");
   break;
  }
 }

 while (1) {};
}
```

# 4 Ground Control

## 4.1 Testing adc reads for using Joystick potentiometers

```c
// Arthur: Mohammed Ibrahim
// Read 4 potentiometers which are 4 channels : THRUT, AILE, RUDD, ELEV
// Acknowledgement: [1] Had to re-write code from start for the adc functions
// which were taken from Rhys thomas
// Potential reason for my code not working : didnt do line 25
// here ADMUX = 2 represents PB3
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>
#include "rfm12.h" // for uplink trasceiver
// #include "rfm12.h" // for downlink transceiver
uint16_t thrust,yaw,pitch,roll;
//****
#define THRUST_TYPE 00 // 0x00 isnt a the actual vallue
#define PITCH_TYPE 01
#define YAW_TYPE 10
#define ROLL_TYPE 11
//****
// initialzie adc
void adc_init()//[1]
{


 // In ADCSRA Enable ADC (set ADEN) and prescaler of 64
 ADCSRA |= _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1);
}
uint16_t adc_read(int n)//[1]
{
 ADMUX = n;// represents PA2
 // start conversion
 ADCSRA |= _BV(ADSC);
 // wait for conversion to complete
 //while(!(ADCSRA & _BV(ADIF))){};
 while(ADCSRA & _BV(ADSC));
 ADC = (ADCH << 8) | ADCL;// [1]
 return ADC;
}
#define BAUD 9600                                   // define baud
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)            // set baud rate value for UBRR

void init_uart1()// initialize UART
{
  //1. set the baud rate, lets configure to 9600;
 // set the baud rate registers Ref: [1],[2]
 UBRR0H = BAUDRATE >> 8;// UBRRnH is 8 bits left
 UBRR0L = BAUDRATE;

  //2. setting up data packet: 8 bits ,no parity 1 stop bit
  // setting 8 bits got to UCSCR register Ref:[3], pg 185 of data sheet

 UCSR0C = _BV(UCSZ00) | _BV(UCSZ01); // 8 bits, USBS1 = 0 for 1 stop bit
```

```

  // note: havnt set up the stop bit in Ref [2] slides
 // 3. from Ref[2] we now enable Transmission and receive n UCSRnB register
 UCSR0B = _BV(TXEN0) | _BV(RXEN0);

}
// transmit data function
void uart_transmit( char data)
{
 while(!( UCSR0A & _BV(UDRE0) ) ); // data register enable bit is 1 if tx buffer is empy
 // if its 1 we load data onto UDR- Uart Data Register(buffer)
 UDR0 = data;
}

void send_string(char *str)
{
 int i;
 for( i = 0; str[i]; i++) uart_transmit(str[i]);
}//***************
int main()
{
 adc_init();
 uplink_rfm12_init();// initialize rfm12 transceiver
 downlink_rfm12_init();
 sei();// enable the ISR in the rfm12.h
 while(1)// main forver loop
 {
  thrust = adc_read(0);// 10 bit value
  // split the 10 bit to 2 bits()
  // transmit it - rfm12_tx() and rfm_tick()
  // rfm12_tx() - fills the tx buffer and transmits the 8-bit type and 8-bit data
  // rfm_tick() checks if channel is free to send next data packet
  // potential delay for sync
  yaw = adc_read(1);
  // split the 10 bit to 2 bits()
  // transmit it - rfm12_tx() and rfm_tick()
  // potential delay for sync
  pitch = adc_read(2);
  // split the 10 bit to 2 bits()
  // transmit it - rfm12_tx() and rfm_tick()
  // potential delay for sync
  roll = adc_read(3);
  // split the 10 bit to 2 bits()
  // transmit it - rfrm12_tx() and rfm_tick()
  //_delay_ms(100); ptoential delay to worry about ater
  // this base station code needs to do reception as well in this while loop
  // functions for down link transceiver

  if (rfm12_rx_status() == STATUS_COMPLETE)// if receiveing data is done, then read the buffer
  {
   uint8_t channel_type = rfm12_rx_type(); // read the 8- bit type
   uint8_t channel_data = rfm12_rx_buffer();// read the 8 - bit data
   // important to clear the receiver buffer
   rfm12_rx_clear();
   // then re-obtain the 10-bit data from the 8-bit packet and 8- bit type
   // joel's code bit
   //look at the data type and identify which channel it is
   char ch[20];
   switch (channel_type)
   {
    case THRUST_TYPE: {
         sprintf(ch,"THRUST = %d",channel_data);
         send_string(ch);
         }
        break;
    case PITCH_TYPE: {
         sprintf(ch,"PITCH = %d",channel_data);
         send_string(ch);
         }
        break;
    case ROLL_TYPE: {
         sprintf(ch,"ROLL = %d",channel_data);
         send_string(ch);
         }
        break;
    case YAW_TYPE: {
         sprintf(ch,"ROLL= %d",channel_data);
         send_string(ch);
         }
        break;
   }
  }

 }
}
```

## 4.2 Testing User Interface code in the base station controler

```
// Code for entering PID constants values on putty and then
// step 1: get them ton display on screen
// step 2: Once step 1 is done get it to display then figure out how to convert them to 8 bit values that show a resolution between 0 and 1
// and display them
#include <avr/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <util/delay.h>
#include <avr/interrupt.h>
//#include "debug.h"
#define BAUD 9600                                   // define baud
#define BAUDRATE ((F_CPU)/(BAUD*16UL)-1)            // set baud rate value for UBRR
#define CHAR_MAX 6
//********** from servo.c

#define PWM_DUTY_MAX 240
#define PWM_DUTY_MIN 0
#define PWM_PRESCALER 8UL
#define PWM_FREQUENCY 50
#define PWM_OFFSET 0
//**********

/*Includes usart.h header file which defines different functions for USART. USART header file version is 1.1*/
void adc_init()//[1]
{

 // In ADCSRA Enable ADC (set ADEN) and prescaler of 64
 ADCSRA |= _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1);
}
void init_pwm(void)
{
    /* TIMER 2 */
    DDRD |= _BV(PD6); /* PWM out */
    DDRD |= _BV(PD7); /* inv. PWM out */


    TCCR2A = _BV(WGM20) | /* fast PWM/MAX */
      _BV(COM2A1); /* A output */
    TCCR2B = _BV(CS21)  |
             _BV(CS22);   /* 1/256 prescaling */
}
uint16_t adc_read(int n)//[1]
{
 ADMUX = n;// represents PA2
 // start conversion
 ADCSRA |= _BV(ADSC);
 // wait for conversion to complete
 //while(!(ADCSRA & _BV(ADIF))){};
 while(ADCSRA & _BV(ADSC));
 ADC = (ADCH << 8) | ADCL;// [1]
 return ADC;
}
void pwm_duty(uint8_t duty)// from servo.c
{
 duty = duty > PWM_DUTY_MAX ? PWM_DUTY_MAX : duty;
 duty = duty < PWM_DUTY_MIN ? PWM_DUTY_MIN : duty;
     //printf("\nPWM=%3u  ==>  ", duty);
 OCR2A = duty;
}
void init_uart1()// initialize UART
{
  //1. set the baud rate, lets configure to 9600;
 // set the baud rate registers Ref: [1],[2]
 UBRR0H = BAUDRATE >> 8;// UBRRnH is 8 bits left
 UBRR0L = BAUDRATE;

  //2. setting up data packet: 8 bits ,no parity 1 stop bit
  // setting 8 bits got to UCSCR register Ref:[3], pg 185 of data sheet

 UCSR0C = _BV(UCSZ00) | _BV(UCSZ01); // 8 bits, USBS1 = 0 for 1 stop bit

  // note: havnt set up the stop bit in Ref [2] slides
 // 3. from Ref[2] we now enable Transmission and receive n UCSRnB register
 UCSR0B = _BV(TXEN0) | _BV(RXEN0) | _BV(RXCIE0);// enable rx interrupt

}
// transmit data function to transmit to the screen
void uart_transmit( char data)
{
 while(!( UCSR0A &  _BV(UDRE0) ) ); //  data register enable bit is 1 if tx buffer is empy
 // if its 1 we load data onto UDR- Uart Data Register(buffer)
 UDR0 = data;
}
// reveive data function to receive values entered on screen
// char uart_receive()
// {
//  if (!(UCSR0A &  _BV(RXC0) ))// if there is unread data in the receive whihc needs to be read
//   return UDR0;
//  else
//   return NULL;
// }
void send_string(char *str)
{
```

```c
 int i;
 for( i = 0; str[i]; i++) uart_transmit(str[i]);
}//***************
char k[CHAR_MAX];
volatile uint8_t counter = 0;// itss uint8_t for size saving
volatile uint8_t pid_enter_check = 0; // used to ensure number(eg: 10.230) is not passed to k[] when asked to enter p or i or d
volatile char pid;
char buff[10];
ISR(USART0_RX_vect)
{
 char temp[CHAR_MAX];
 temp[counter] = UDR0;
 if (temp[counter] == 'p' || temp[counter] == 'i' || temp[counter] == 'd')
 {
  pid = temp[counter];
  uart_transmit(pid);// print it to see what we enter
 }
 else if (temp[counter] == 'x')
 {
  counter = 0;//reset the counter to zero so that u can start re-writing to the k[] array
  send_string("\n\r Re-enter K value: ");
 }
 else if ( (pid_enter_check == 1) && (temp[counter] > 47 || temp[counter] < 58) )
 {
  uart_transmit(temp[counter]);
  send_string("\n\r p or i or d pls!: ");
  counter = 0;// ensure buffer isnt filled with these numbers by re-setting this counter
  pid_enter_check = 0;
 }
 //  ASCII : 47 < x < 75 corresponds to integers 0-9 and x = 46 whihc is a 'dot' which we EXCLUDE in this condition to DEAL with the "other characters" apart fron 'p' or 'i' or 'd'
 else if ( (temp[counter] < 47 || temp[counter] > 57) && (temp[counter] != 46) ) // if what you entered are letters like 'l' or 'z' etc. when asked to enter p or i or d and
 {
  send_string("\n\r Please enter p or i or d: "); // ask to re enter p or i or d
  counter = 0;// BUT if numbers were ented when asked to enter p or i or d
  // re-write the the k[] buffer to get rid of those numbers entered
 }
 else // if numbers or dots were entered
  {
   k[counter] = temp[counter];// can store digits or dots and incremet counter
   uart_transmit(k[counter]);// print it to see what we enter
   counter++;// incremet to get next digit
  }
    // do an else in case its an invalid character entery other than i or p or d

 // ki[counter] = UDR0;// read the
 // //send_string("\n\rin ISR");
 // uart_transmit(ki[counter]);
 // counter++;
 // //sprintf(buff,"%d",counter);
 // //send_string(buff);

}

int main()
{
 init_uart1();
 init_pwm();
 adc_init();
 //init_debug_uart0();
 sei();// enable global interrupt
 char ch[60];
 float f, f_temp;
 uint16_t ten_bit;
 send_string("\n\r Enter a K type (p or i or d): ");
 pid_enter_check = 1;// always set this variable to 1 when asked to enter p or i or d
 uint16_t adc_value;
 uint8_t pwm_value;

 while(1)
 {

  //receive_string(ki);
  //ki[i] = uart_receive();
  adc_value = adc_read(0);
  pwm_value = (uint8_t) (adc_value/4) + PWM_OFFSET;
  pwm_duty(pwm_value);
  // we are having a servio as way of checking if this code doesnt block the flow of this while loop significantly
  if (pid == 'p' || pid == 'i' || pid == 'd')
   {
    send_string("\n\r Enter k value (press x to re-enter): ");
    // BEFORE NULLING IT transmit it!!!!!!!!!!!!!!!!
    //**************
    //**************
    pid = NULL;
   }

  else if (counter == CHAR_MAX)// char bufer is ready to transmit
  {
   counter = 0;// back to zero
   f = atof(k);// convert it to float
   f_temp = f*100; //
   if (f_temp > 1023)// check if number not withing range
   {
    send_string("\n\r Error: k value is not in range!");
    send_string("\n\r Enter a K type (p or i or d): ");
    continue;
   }
   ten_bit = (uint16_t) (f_temp+0.5);
   sprintf(ch,"\n\r 10-bit dec value: %d",ten_bit);
   send_string(ch);
   // transmit its
   send_string("\n\r Enter a K type (p or i or d): ");
   pid_enter_check = 1;// always set this variable to 1 when asked to enter p or i or d
  }
  else
   continue;


 }
}
```

# 5 Chassis Design

## 5.1 Servo code for hook

```c
/*
* Pointless little servo test program
* Controlled by IR sensor
* ADC pin - PA0
* PWM pin - PD7
* RXD pin - PD0 Orange
* TXD pin - PD1 Yellow
*/



#include <avr/io.h>
#include <util/delay.h>
#include "debug.h"

#define PWM_DUTY_MAX 240
#define PWM_DUTY_MIN 0
#define PWM_PRESCALER 8UL
#define PWM_FREQUENCY 50
#define PWM_OFFSET 0

void init_pwm(void);
void pwm_duty(uint8_t duty);

void init_adc(void);
uint16_t adc_read(void);

int main (void)
{

 init_pwm();
 init_adc();
 init_debug_uart0();

 uint16_t adc_value;
 uint8_t pwm_value;

 while(1)
 {
  adc_value = adc_read();
  pwm_value = (uint8_t) (adc_value/4) + PWM_OFFSET;
  pwm_duty(pwm_value);
  _delay_ms(100);
 }
}



void init_pwm(void)
{
    /* TIMER 2 */
    DDRD |= _BV(PD6); /* PWM out */
    DDRD |= _BV(PD7); /* inv. PWM out */


    TCCR2A = _BV(WGM20) | /* fast PWM/MAX */
      _BV(COM2A1); /* A output */
    TCCR2B = _BV(CS21)   |
             _BV(CS22);    /* 1/256 prescaling */
}

void pwm_duty(uint8_t duty)
```

```c
{
 duty = duty > PWM_DUTY_MAX ? PWM_DUTY_MAX : duty;
 duty = duty < PWM_DUTY_MIN ? PWM_DUTY_MIN : duty;
     printf("\nPWM=%3u  ==>  ", duty);
 OCR2A = duty;
}

void init_adc (void)
{
    /* REFSx = 0 : Select AREF as reference
     * ADLAR = 0 : Right shift result
     *  MUXx = 0 : Default to channel 0
     */
    ADMUX = 0x00;
    /*  ADEN = 1 : Enable the ADC
     * ADPS2 = 1 : Configure ADC prescaler
     * ADPS1 = 1 : F_ADC = F_CPU / 64
     * ADPS0 = 0 :       = 187.5 kHz
     */
    ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1);
}


uint16_t adc_read(void)
{
     uint16_t adc_in;

     /* Start single conversion */
     ADCSRA |= _BV ( ADSC );
     /* Wait for conversion to complete */
     while ( ADCSRA & _BV ( ADSC ) );
     adc_in = ADC;

     printf("ADC=%4d", adc_in);

return adc_in;
}
```

## 5.2 Laser printing design