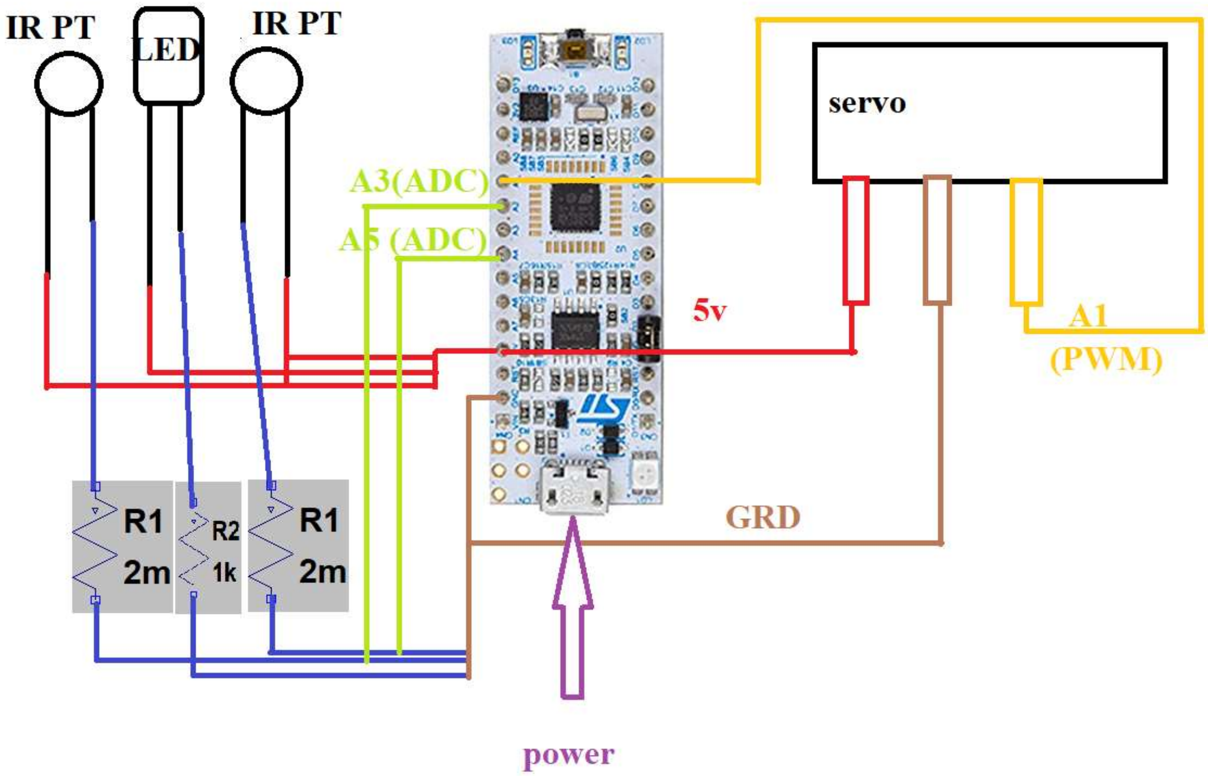
Kunal Mukherjee

2/15/19

Dr. Marc Mitchell

Project 1

Project Schematic



The physical design and the software coding were not the most difficult part of the project. The most difficult part of the project was signal conditioning. The software part has some minor issues as I believe using sequencing the adc channel and using interrupt would be a better design choice as well as optimize the code. I manually had to turn the adc off and on, everything. I think that wastes processor clock cycle and dissipates more energy than necessary. In hindsight, I spent a lot of time looking at the wrong port a I was going by what was in the physical body of the board rather than looking at the pin out in the documentation. So, my sequencing code might have worked but since my adc was connected to the physical wrong channel, the pin was not reading any input.

The signal conditioning was extremely empirical and constitutes majority of the design choices. For example, to get my tracking to work, I had to pass over 150 mA current through the LED, for the IR to reflect back with such intensity that is being detected by the IR transistors. I also connected the IR transistor directly to +5V and the current limiting resister was connected to the ground. In the way, when the IR transistor detects no IR, it shows around ~600, but once it detects the IR then the adc value goes up because now there would be a potential difference as current will pass through IR transistor. I also had to use 2 MΩ resistor to make the device extremely sensitive so that it can detect deflected IR. I started with 500 Ω and with a bad decision, and slowly I increased the resistor values and choosing 2 M Ω was a desperation choice, and it worked as the previous values were showing me ~.50V difference when it was detecting deflected IR. But, 2 MΩ has almost ~2.1 V increase when it detects deflected IR.

The project could have been more accurate, and tracking could have been better, if I had access to more powerful IR emitter or use many LED together so that the IR reflected would be strong enough for the transistor to detect. It could also be better if I had access to a 2 MΩ potentiometer, so that I could test and figure out which resistor value for the IR transistor works best. The device steps to move also took a long time, so the steps for each move should be dynamically changed depending on the object it is tracking. Also, the IR sensor were customized for a specific backlight, and it will have hard time tracking something in a different lighting condition. An algorithm, that looks at the background gradient and dynamically changes the background number which predicts the IR reflected number threshold. Ultimately, to make this project really well, all the #define number needed to be a volatile define, so that during run-time we can change the values dynamically so that this device can be optimized for any condition.

```c
//Kunal Mukherjee;
//2/10/2019
//Proj 1

//the include files
#include "stm32l432.h"

//ADC that takes input with a channel
void ADC_Init(int channel);
void GPIO_Init(void);
void ADC_CLK_GPIO(void);

#define DELTA 130
#define MAX 5000 //2.5 ms = 5000/40000 * 20
#define MIN 1000// 0.5 ms = 1000/40000 * 20
#define MIDDLE 3000 //1.5 ms = 3000/40000 * 20

#define RIGHT_EMPTY 680 //at room light this is what the RIGHT_LED read
                        //anything greater means that an IR has replected back
                        // so it is seeing something
#define LEFT_EMPTY 668  //

#define DELAY 250 // wait for that may counts before resuming work

int main()
{

  //assign the variables
  unsigned int i, j, tmr2, choice, diff;
  int right_sensor, left_sensor;

  GPIO_Init();
  ADC_CLK_GPIO();

  //initilization of the variables
  tmr2 = MIDDLE;
  right_sensor = left_sensor = 0;

  while(1)
  {
    ADC_Init(8); //setup adc of channel 8
    ADC_CR |= (1 << 2); //start adc regular conversion
    while((ADC_ISR & (1 << 2)) == 0); // end of regular sequence flag
    left_sensor = ADC_DR & 0xFFF; //only look at 12 bit
    ADC_CR |= (1 << 1); //turn ADC off

    for (i = 0; i < DELAY; i++); //delay

    ADC_Init(10); //setup adc for channel 10
    ADC_CR |= (1 << 2); //start adc regular conversion
    while((ADC_ISR & (1 << 2)) == 0); // end of regular sequence flag
    right_sensor = ADC_DR & 0xFFF; //only look at 12 bit
    ADC_CR |= (1 << 1); //turn ADC off

    for (i = 0; i < DELAY; i++); //delay

    //getting the diff betwen the sensors
    if (right_sensor > left_sensor)
    {
        diff = right_sensor - left_sensor;
    }else{
        diff = left_sensor - right_sensor;
    }

    //nothing to follow
    //so the finding algorithm
    if (((right_sensor <= RIGHT_EMPTY)) &&
        ((left_sensor <= LEFT_EMPTY) ))
    {
      if (choice == 0)
      {
        tmr2 += DELTA;
```

```c
          if (tmr2 > MAX)
          { choice = 1; tmr2 = MAX;}
        }
        else
        {
          tmr2 -= DELTA;
          if (tmr2 < MIN)
          {choice = 0; tmr2 = MIN;}
        }
      }

      //sensors sensed something
      else
      {
         if (diff < 70) //the object is in the middle
         {
            //do nothing, item found
         }

        else {
            if (right_sensor > left_sensor) //right sensor has sensed something
            {
              tmr2 += DELTA; //add some movement to right

              if (tmr2 > MAX) //if max then do not move from the right
              { tmr2 = MAX; }
            }else{ //left sensor has sensed something
              tmr2 -= DELTA;  // add some movement to the left

              if (tmr2 < MIN) //if mov then do not move from the left
              { tmr2 = MIN; }

            }
        }
      }

      //code to check if -180+180 motion is being read
      /*if (choice == 0)
        {
          tmr2 += DELTA;
          if (tmr2 > MAX)
            choice = 1;
        }else{
          tmr2 -= DELTA;
          if (tmr2 < MIN)
            choice = 0;
        }*/

      //enter the new high time value to the CCR2 reg
      TIM2_CCR2 = tmr2; //scale the value

      for (i = 0; i < 10000; i++){for (j = 0; j < 5; j++);}; //delay
    }
   }

  void GPIO_Init(void)
  {
    //clock initilaizations
    RCC_AHB2ENR  |= (1 << 0);  //set the GPIOA clk
    RCC_APB1ENR1 |= (1 << 0); //TIM2 en from APB1 peri clk enb reg

    //GRIO setup
    GPIOA_MODER   &= ~(3 << (2 * 1)); //clear the GPIOA mode bits
    GPIOA_MODER   |=  (2 << (2 * 1)); //set port a1 is alternate 10
    GPIOA_OTYPER  &= ~(1 << (1 * 1)); //open drain for a1
    GPIOA_OTYPER  |=  (1 << (1 * 1)); //open drain for a1
    GPIOA_OSPEEDR &= ~(3 << (2 * 1)); //high speed output
    GPIOA_OSPEEDR |=  (2 << (2 * 1)); //high speed output
    GPIOA_AFRL    |=  (1 << (4 * 1)); //alt func 1, port pin 5,
                                     //control bit are 4 bit wide

    TIM2_CR1 |= (1 << 7); //ARPE: Auto-reload preload enable
```

```c
  TIM2_PSC = 1;           //PSC set to 2 = 1 + 1
  TIM2_ARR = 40000;       //50 Hz = 20  = ms; 4MHz/2 = 2MHz; 2MHz/40000 = 50Hz
  TIM2_CCMR1 |= 0x6800; //Channel 2;
                          //bit 11: OC2PE: Output compare 2 preload enable
                          //0110: PWM mode 1 - In upcounting,
                          //channel 1 is active as long as
                          //TIMx_CNT<TIMx_CCR1else inactive.
  TIM2_CCER |= (1 << 4);//CC1E: Capture/Compare 2 output enable.
  TIM2_CCR2 |= MIDDLE;  //CCR2 is the value to be loaded in the actual
                          //capture/compare 2 register (preload value).
  TIM2_EGR |= (1 << 0); //UG: update event
  TIM2_CR1 |= (1 << 0); //CEN: counter enabled
}

void ADC_CLK_GPIO(void)
{
  //set up the adc clock
  RCC_AHB2ENR  |= (1 << 13); //set ADC clk

  //adc GPIO Setup
  //mode default to analog for PA3 & PA5
  GPIOA_PUPDR &= ~(3  << (2 * 3)); //PA3 to no pull or down
  GPIOA_PUPDR &= ~(3  << (2 * 5)); //PA5 to no pull or down
}


void ADC_Init(int channel)
{
  ADC_CR &= ~(1 << 0); //disable ADC

  int i;  //a counter for .5 us

  ADC_CR &= ~(1 << 29); //deep power mode cleared
  ADC_CR |= (1 << 28); //set voltage reg

  for (i=0; i <10000; i++);//wait for .5us

  ADC_CCR |= (1 << 22); //VREF ENAB
  ADC_CCR |= (1 << 16); //HCLK/1 (Synchronous clock mode) enb

  ADC_ISR |= (1 << 0); //ADC ready
  ADC_CR |= (1 << 0); //ENB ADC

  while ((ADC_ISR & (1 << 0)) == 0); //wait till ADC is ready

  ADC_SQR1 &= ~(31 << 6);
  ADC_SQR1 |=  (channel << 6);//CH8 for A3 or CH10 for A5

  ADC_CFGR |= (1 << 16); //DISCEN: Discontinuous mode for regular channels
}
```