|  |  |
| --- | --- |
| My little segway  Project Report | 제출정보  교 과 목 : 임베디드 시스템 설계  담당교수 : 임 성 수 교수님  학 과 : 컴퓨터 공학부  제 출 일 : 2015. 12. 14  제출인  20133242 이두나 |

my Little Segway

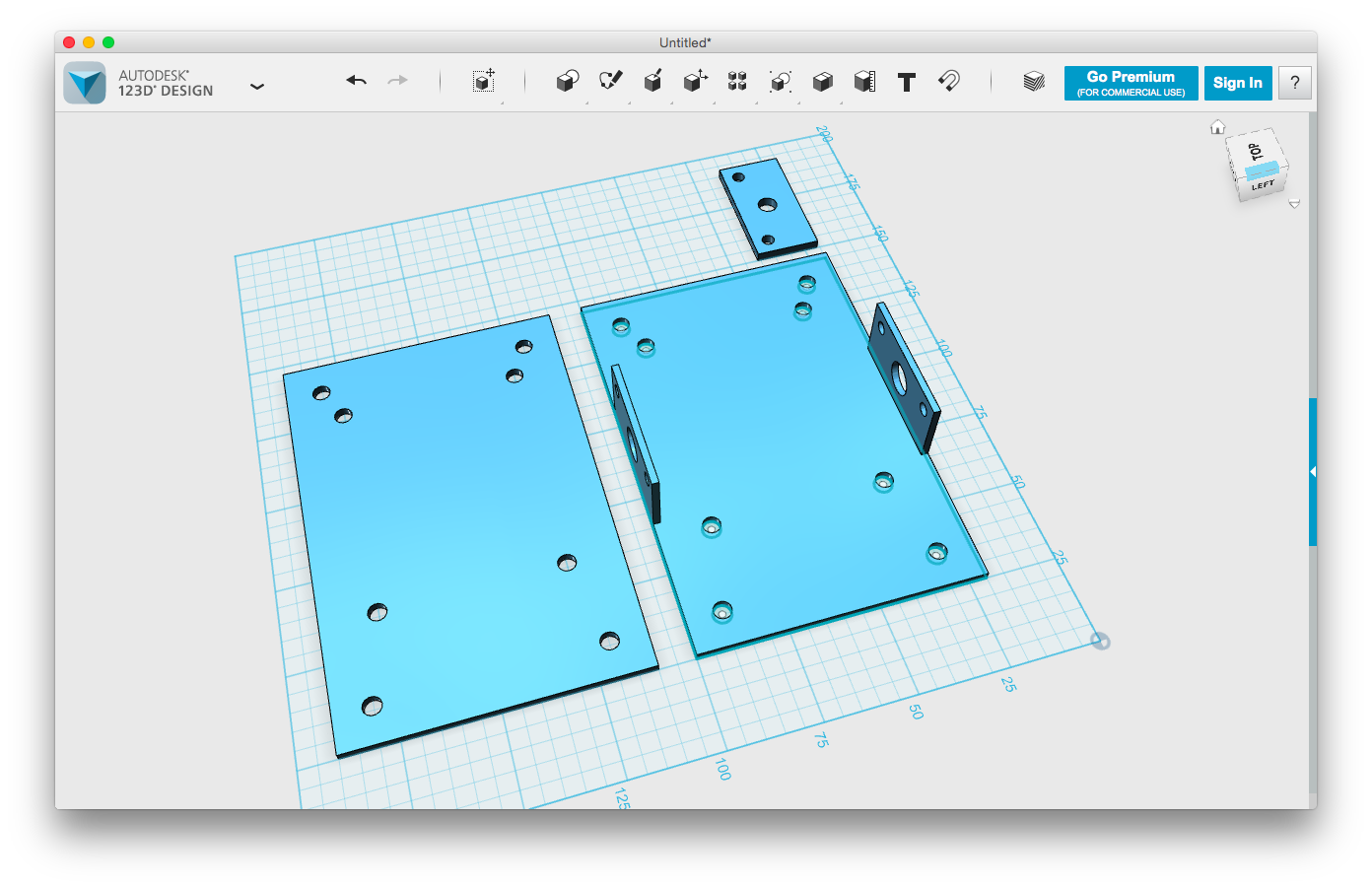
# Project description

The Segway is a two-wheeled, self-balancing, battery-powered electric vehicle used as a convenient way to travel around towns and cities. In this project, I made it small. However, if you use the better motor you can make the real Segway possible to ride.

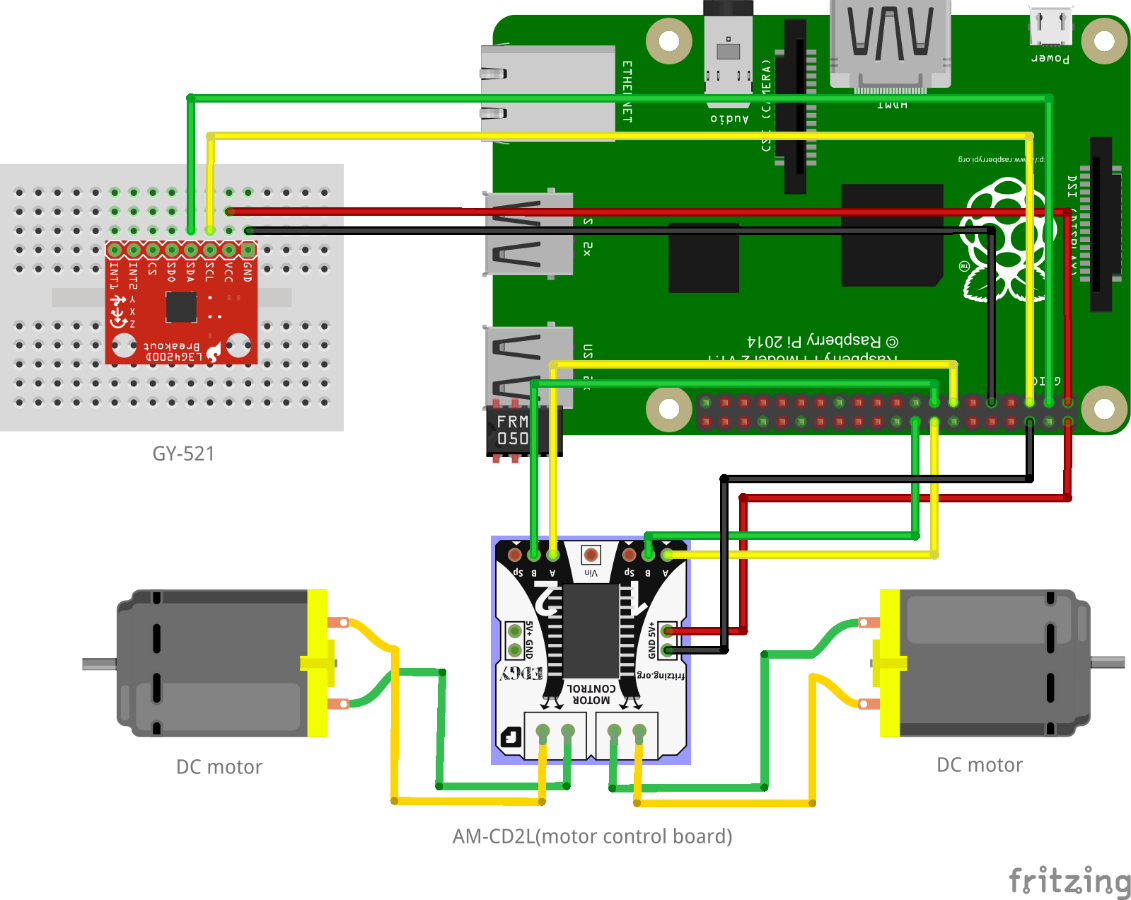
By following this project you will learn how to access 6-axis motion sensor, motor control, PID control and basic automatic control theorem.

# System Overvie

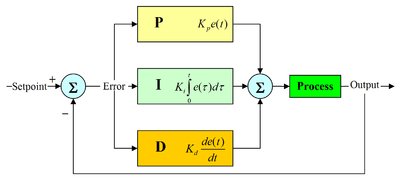
1. Design View



1. Schematic Design



1. Basics Software



# environment

1. OS : Raspbian
2. Tool : 123D design(3D design program), Cura(slicer program), Friaing(for schematic Design), vim
3. SW packages : wiringPi, pwm

# source code

**<motors.c>**

#include <stdio.h>

#include <stdlib.h>

#include <wiringPi.h>

#include <softPwm.h>

#define RANGE 100

#define INITIAL\_VALUE 0

void init\_motors()

{

wiringPiSetup();

// initialize left motor (wiring pin 2 & 3)

softPwmCreate(2, INITIAL\_VALUE, RANGE);

softPwmCreate(3, INITIAL\_VALUE, RANGE);

// initialize right motor (wiring pin 4 & 5)

softPwmCreate(4, INITIAL\_VALUE, RANGE);

softPwmCreate(5, INITIAL\_VALUE, RANGE);

}

void stop\_motors()

{

pinMode(2, OUTPUT);

pinMode(3, OUTPUT);

digitalWrite(2, LOW);

digitalWrite(3, LOW);

pinMode(4, OUTPUT);

pinMode(5, OUTPUT);

digitalWrite(4, LOW);

digitalWrite(5, LOW);

// printf("motor stopped!!\n");

// exit(1);

}

double left\_speed;

double right\_speed;

void motors(double speed, double left\_offset, double right\_offset)

{

// to come to me, drive pin 3&5 to some power

// to away from me, drive pin 2&4 to some power

// 2:LM-, 3:LCE

// 4:RM-, 5:RCE

// put M+ high & M- low will come to me

// put M+ low & M- High will away from me

left\_speed = speed + left\_offset;

right\_speed = speed + right\_offset;

// left motor

if (left\_speed < 0) {

softPwmWrite(2, (int) -left\_speed);

softPwmWrite(3, 0);

}

else

if (left\_speed > 0) {

softPwmWrite(3, (int) left\_speed);

softPwmWrite(2, 0);

}

// right motor

if (right\_speed < 0) {

softPwmWrite(5, (int) -right\_speed);

softPwmWrite(4, 0);

}

else

if (right\_speed > 0) {

softPwmWrite(4, (int) right\_speed);

softPwmWrite(5, 0);

}

}

**<mySegway.c>**

#include <wiringPiI2C.h>

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <sys/time.h>

// PID parameters

double Kp = 2.5; // 2.5

double Ki = 0.8; // 1.0

double Kd = 8.0; // 8.0

double K = 1.9\*1.12;

// Complimentary Filter parameters

double K0 = (double) 0.98;

double K1 = (double) 0.02;

int fd;

int acclX, acclY, acclZ;

int gyroX, gyroY, gyroZ;

double accl\_scaled\_x, accl\_scaled\_y, accl\_scaled\_z;

double gyro\_scaled\_x, gyro\_scaled\_y, gyro\_scaled\_z;

double gyro\_offset\_x, gyro\_offset\_y;

double gyro\_total\_x, gyro\_total\_y;

double gyro\_x\_delta, gyro\_y\_delta;

double rotation\_x, rotation\_y;

double last\_x, last\_y;

struct timeval tv, tv2;

unsigned long long timer, t;

double deltaT;

int read\_word\_2c(int addr)

{

int val;

val = wiringPiI2CReadReg8(fd, addr);

val = val << 8;

val += wiringPiI2CReadReg8(fd, addr+1);

if (val >= 0x8000)

val = -(65536 - val);

return val;

}

double dist(double a, double b)

{

return sqrt((a\*a) + (b\*b));

}

double get\_y\_rotation(double x, double y, double z)

{

double radians;

radians = atan2(x, dist(y, z));

return -(radians \* (180.0 / M\_PI));

}

double get\_x\_rotation(double x, double y, double z)

{

double radians;

radians = atan2(y, dist(x, z));

return (radians \* (180.0 / M\_PI));

}

void read\_all()

{

acclX = read\_word\_2c(0x3B);

acclY = read\_word\_2c(0x3D);

acclZ = read\_word\_2c(0x3F);

accl\_scaled\_x = acclX / 16384.0;

accl\_scaled\_y = acclY / 16384.0;

accl\_scaled\_z = acclZ / 16384.0;

gyroX = read\_word\_2c(0x43);

gyroY = read\_word\_2c(0x45);

gyroZ = read\_word\_2c(0x47);

gyro\_scaled\_x = gyroX / 131.0;

gyro\_scaled\_y = gyroY / 131.0;

gyro\_scaled\_z = gyroZ / 131.0;

}

unsigned long long getTimestamp()

{

gettimeofday(&tv, NULL);

return (unsigned long long) tv.tv\_sec \* 1000000 + tv.tv\_usec;

}

double constrain(double v, double min\_v, double max\_v)

{

if (v <= min\_v)

return (double)min\_v;

else if (v >= max\_v)

return (double)max\_v;

else

return (double)v;

}

double GUARD\_GAIN = 100.0;

double error, last\_error, integrated\_error;

double pTerm, iTerm, dTerm;

double angle;

double angle\_offset = 2.0; //1.5

double speed;

void pid()

{

error = last\_y - angle\_offset;

pTerm = Kp \* error;

integrated\_error = 0.95\*integrated\_error + error;

iTerm = Ki \* integrated\_error;

dTerm = Kd \* (error - last\_error);

last\_error = error;

speed = constrain(K\*(pTerm + iTerm + dTerm), -GUARD\_GAIN, GUARD\_GAIN);

}

int main()

{

init\_motors();

delay(200);

fd = wiringPiI2CSetup (0x68);

wiringPiI2CWriteReg8 (fd,0x6B,0x00);//disable sleep mode

// printf("set 0x6B=%X\n",wiringPiI2CReadReg8 (fd,0x6B));

timer = getTimestamp();

deltaT = (double) (getTimestamp() - timer)/1000000.0;

read\_all();

last\_x = get\_x\_rotation(accl\_scaled\_x, accl\_scaled\_y, accl\_scaled\_z);

last\_y = get\_y\_rotation(accl\_scaled\_x, accl\_scaled\_y, accl\_scaled\_z);

gyro\_offset\_x = gyro\_scaled\_x;

gyro\_offset\_y = gyro\_scaled\_y;

gyro\_total\_x = last\_x - gyro\_offset\_x;

gyro\_total\_y = last\_y - gyro\_offset\_y;

while(1) {

t = getTimestamp();

deltaT = (double) (t - timer)/1000000.0;

timer = t;

read\_all();

gyro\_scaled\_x -= gyro\_offset\_x;

gyro\_scaled\_y -= gyro\_offset\_y;

gyro\_x\_delta = (gyro\_scaled\_x \* deltaT);

gyro\_y\_delta = (gyro\_scaled\_y \* deltaT);

gyro\_total\_x += gyro\_x\_delta;

gyro\_total\_y += gyro\_y\_delta;

rotation\_x = get\_x\_rotation(accl\_scaled\_x, accl\_scaled\_y, accl\_scaled\_z);

rotation\_y = get\_y\_rotation(accl\_scaled\_x, accl\_scaled\_y, accl\_scaled\_z);

// printf("[BEFORE] gyro\_scaled\_y=%f, deltaT=%lf, rotation\_y=%f, last\_y= %f\n", (double)gyro\_scaled\_y, (double)deltaT, (double)rotation\_y, (double) last\_y);

// printf("[1st part] = %f\n", (double) K0\*(last\_y + gyro\_y\_delta));

// printf("[2nd part] = %f\n", (double) K1\*rotation\_y);

last\_x = K0 \* (last\_x + gyro\_x\_delta) + (K1 \* rotation\_x);

last\_y = K0 \* (last\_y + gyro\_y\_delta) + (K1 \* rotation\_y);

// printf("[AFTER] gyro\_scaled\_y=%f, deltaT=%lf, rotation\_y=%f, last\_y=%f\n", (double)gyro\_scaled\_y, (double)deltaT, (double)rotation\_y, (double) last\_y);

if (last\_y < -60.0 || last\_y > 60.0)

stop\_motors();

pid();

printf("%lf\t%lf\t%lf\t%lf\t%lf\n", error, speed, pTerm, iTerm, dTerm);

motors(speed, 0.0, 0.0);

delay(10);

}

stop\_motors();

return 0;

}