<조선해양공학계획>

Term Project 최종보고서

-VLCC를 기준선형으로 초음파센서를 이용한 모형선의 자동제어-

7조 Renovatia

김두용 김영현 노재욱 심훈섭 이주현

1. 선체부 설계

본 장에서 선체부 설계를 크게 두 단계로 나누어 기술 하였다. 먼저 Proposal 단계에서의 기준선 선정과 개략 경하 중량의 추정 및 그에 따른 주요 치수의 결정 과정을 기초 설계로 칭해 기술하였고, 개략적으로 결정된 주요 치수를 토대로 EzShip 을 활용한 선형/구획 설계 및 세부 경하 중량의 추정과 그에 따른 설계 선형의 확정 과정을 상세 설계로 칭해 기술한 것이 두 번째 단계이다.

먼저 본 조의 설계에서 고려된 과제의 요구조건은 다음과 같았다.

- 정해진 재화중량(6kg)을 실을 수 있어야 함
- 전장은 1m 내외를 권장
- 선체의 소재는 자유

  -Foam board, 나무, 과학상자
- 구동(추진)시스템 자유

  -Propeller, 외륜선
- 선형 선택 자유- 단순한 선형을 선택
- 학생들의 창의성이 중요

1.1. 기초설계

1.1.1. 기준선의 선정

이번 프로젝트에서 우리에게 주어진 과제는 재화중량 6kg을 싣고

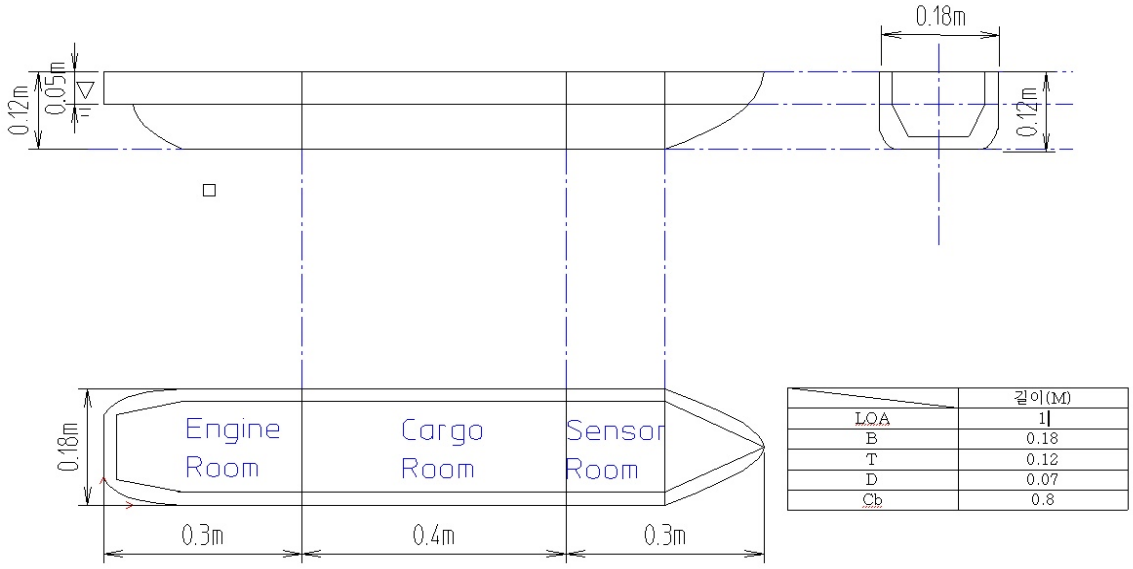
1) 벽면과의 일정거리를 유지하는 자동제어 시스템의 구축

2) 유사시 리모트 컨트롤러로의 전환을 통한 선박 조종성 유지

로서 그 프로젝트의 초점이 창의적 선형 설계에 있기 보다는 선박의 자동제어능력에 있다고 할 수 있다. 따라서 VLCC나 대형 화물선, 즉 방형계수가 커서 선회능력이 좋고 화물을 싣기 용이한 배들을 기준선으로 채택하는 것이 합리적 판단이라 생각하였다.

우리 Renovatia에서는 본 프로젝트의 수행과제에 부합하는 모형 선박을 제작하기 위해 다음 기준선에 대해 조사해 보았다. VLCC ; VLCC는 모형선박을 건조하기에 가장 적합한 선형을 띄고 있다. 선회성능이 좋아서 장애물을 피해야 하는 이번 프로젝트의 목적에 맞을 뿐만 아니라 방형계수가 적당히 커서 재화중량 6KG을 싣는데 어려움이 없다. 선형자체의 창의성은 떨어진다고 생각되지만 이번 프로젝트의 주목적이 제어부의 설계,제작인 만큼 선형에서의 부담을 최소화 하기위해 VLCC를 기준선으로 결정하기로 했다.

제안 당시 개략적으로 주요치수를 결정한 설계선을 기준으로 선형 및 구획 설계를 하였는데 그 개략 배치도는 다음과 같았다.



<그림 1> 개략배치도(Proposal)

1.1.2. 주요치수의 결정

1차 설계 단계에서 개략 경하 중량의 추정은 주요 치수(L, B, D, T)를 결정하기 위한 가장 핵심적인 요소이다. 선형설계시스템인 Ezship을 통해 정확한 경하중량 추정이 이루어지기 전까지 1차 주요치수의 결정을 위해 비교적 정확하게 이루어져야 한다.

이 과정은 다음과 같이 이루어진다

- L, B, T, D, Cb 의 선정
- 위 치수에 따른 만재배수량 계산

- 경하중량의 추정
- DWT = 만재배수량 – 경하중량 의 관계식에서 DWT가 요구 재화중량인 6kg 을 만족하는지 확인
- 만족하지 않으면 다시 처음 단계에서 주요수치 수정 후 동일 과정 반복을 통한 주요 치수의 확정

이 과정에서 어려운 부분은 경하중량의 추정이다. 주요 치수의 수정에 따른 선각 및 내부 부재의 중량의 변화를 계속적으로 계산하여야 하기 때문이다. 따라서 이 부분을 스프레드 시트 프로그램을 통해 L, B, D, T, Cb 입력값을 받아 경하중량을 개략적으로 계산할 수 있게 한다면 주요 치수를 재화중량 6kg을 만족시키기 위해 여러 번 수정을 하더라도 그 때마다 새로이 경하중량을 추정할 필요 없이 용이하게 주요 치수를 결정할 수 있게 된다.

따라서 본 조에서는 마이크로소프트사의 엑셀을 이용하여 L, B, D, T, Cb 및 각종 부재의 크기, 개수 등을 입력 받아서 경하중량을 추정하게 하게끔 하는 프로그램을 작성하였다. 경하중량을 계산하는 과정은 다음과 같았다.

1.1.3. 경하중량의 계산

모형선의 경하중량 중 선체 중량을 제외한 추진부 및 제어부 중량은 주요 치수의 소폭 변화에 대해 무관하게 고정 값으로 간주 할 수 있다. 이는 선체 중량의 계산 후 가산하기만 하면 되는 것이므로 이에 대해서는 나중에 그 목록과 중량만 간략하게 도시하겠다. 선체의 중량은 선각 중량과 내부 부재 중량으로 나누어 계산할 수 있다.

* 선각중량

선각의 중량은 선각 부분을 구성하는 우드락과 그 위에 입혀지는 퍼티와 에폭시의 부피만 알면 밀도를 곱함으로써 얻어진다. 부피를 구할 수 있는 방법이 두 가지가 고안되었는데 이는 다음과 같다.

- 선각의 면적을 구한 후 그 두께를 곱함
- 구각의 부피를 구할 때 쓰는 방법으로 선각의 바깥쪽 면으로 이루어진 입체의 부피에서 선각의 안쪽 면으로 이루어진 입체의 부피를 뺌

본 조에서는 후자의 방법을 취했는데 이는 좀 더 간단하면서도 그 정확성에 있어서 전자의 그것과 비해서 차이가 나지 않는다고 판단했기 때문이다.

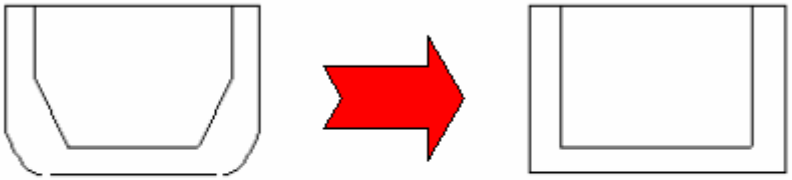
선체 외곽으로 이루어진 전체 부피는 L*B*D*Cb 로 근사적으로 계산해볼 수 있고 이 때 선각의 두께가 d라면 선각 안쪽 부분의 부피는 (L-2d)*(B-2d)*(D-d)*Cb로 계산될 수 있어 이에 그 선각의 밀도만 곱하면 선각의 중량을 쉽게 추정해볼 수 있다.

선체 내부 부재 중량

선체 내부 부재는 크게 세 부분으로 나누어 생각해 볼 수 있다. 선체의 외피를 입히는 부분인 U자형 부재와 그것들을 고정시키는 종방향의 부재(용골에 해당함), 그리고 선체 외곽을 따라 U자형 부재들을 고정시키는 가로방향 부재가 그에 해당한다. 각자를 개략적으로 다음과 같이 추정하였다.

- U자형 부재

중앙평행부는 근사적으로 사각형에 가까우므로 사각형의 면적으로 쉽게 구할 수 있다.



<그림 2> 중앙평행부 면적 근사

- 용골에 해당하는 부재는 3개를 넣는데 이의 부피는 폭을 입력 받아서 우드락의 두께와 배의 전장(LOA)를 곱하여 계산하였다.
- 사이드라인을 따라 감고 도는 가로 방향의 부재는 선체가 방형계수가 큰 육면체에 가까운 선형임을 고려해 마찬가지로 사각형의 면적으로 근사하여 추정하였다.

다음 아래 표는 주요 치수 및 주요 부재의 두께 혹은 폭, 개수 등을 입력 받아 여러 주요 요목의 비 및 만재 배수량, 경하중량 및 그에 따른 재화중량(DW)를 계산해주는 엑셀 프로그램을 작성하여 여러 치수를 입력하여 출력된 화면이다.

<표 1> 주요치수결정

		case 1	case 2	case 3	case 4
LOA	(m)	1.1	1.1	1	1
LBP	(m)	1	1	0.95	0.95

B	(m)	0.2	0.2	0.2	0.18
D	(m)	0.12	0.12	0.12	0.12
T	(m)	0.06	0.06	0.07	0.07
Cb	무차원	0.8	0.8	0.8	0.8
L/B	무차원	5	5	4.75	5.27777777
B/T	무차원	3.33333333	3.33333333	2.85714285	2.57142857
L/D	무차원	8.33333333	8.33333333	7.91666667	7.91666666
B/D	무차원	1.666666667	1.666666667	1.666666667	1.5
					0.15157894
비만계수		0.16	0.16	0.168421053	7
만재배수량	kg	9.6	9.6	10.64	9.576
우드락두께	(mm)	2	2	2	2
퍼티두께	(mm)	0.5	0.5	0.5	0.5
에폭시두께	(mm)	0.5	0.5	0.5	0.5
외곽 부피	m^3	0.02112	0.02112	0.0192	0.01728
내곽 부피	m^3	0.01993492	0.019934925	0.018116045	0.01626746
선각부피	m^3	0.00118507	0.001185075	0.001083955	0.00101253
선각폼보드	kg	0.82955264	0.82955264	0.75876864	0.70877184
에폭시퍼티	kg	0.84436608	0.84436608	0.77231808	0.72142848
선각총중량	kg	1.67391872	1.67391872	1.53108672	1.43020032
격벽개수	개	4	4	3	3
격벽중량		0.1344	0.1344	0.1008	0.09072

U부재폭	m	0.03	0.03	0.03	0.2
U부재 부피	m^3	0.0000252	0.0000252	0.0000252	0.0000352
U부재 개수	개	26	26	26	26
U부재부피		0.0006552	0.0006552	0.0006552	0.0009152
U부재중량		0.412776	0.412776	0.412776	0.576576
용골폭	m	0.03	0.02	0.03	0.03
가로부재폭	m	0.03	0.02	0.03	0.03
용골부피(3개)		0.000198	0.000132	0.00018	0.00018
가로부재부피(3개)		0.0003638	0.0003888	0.0003298	0.000291
용골중량	kg	0.1386	0.0924	0.126	0.126
가로부재중량	kg	0.25466	0.27216	0.23086	0.2037
총부재중량	Kg	0.940436	0.911736	0.870436	0.996996
추진/제어부중량	Kg	1	1	1	1
만재배수량	Kg	9.6	9.6	10.64	9.576
개략경하중량	Kg	3.61435472	3.58565472	3.40152272	3.42719632
재화중량	Kg	5.98564528	6.01434528	7.23847728	6.14880368

이중 기관부 중량은 1kg으로 고정인데 이의 상세 내역은 다음과 같다.

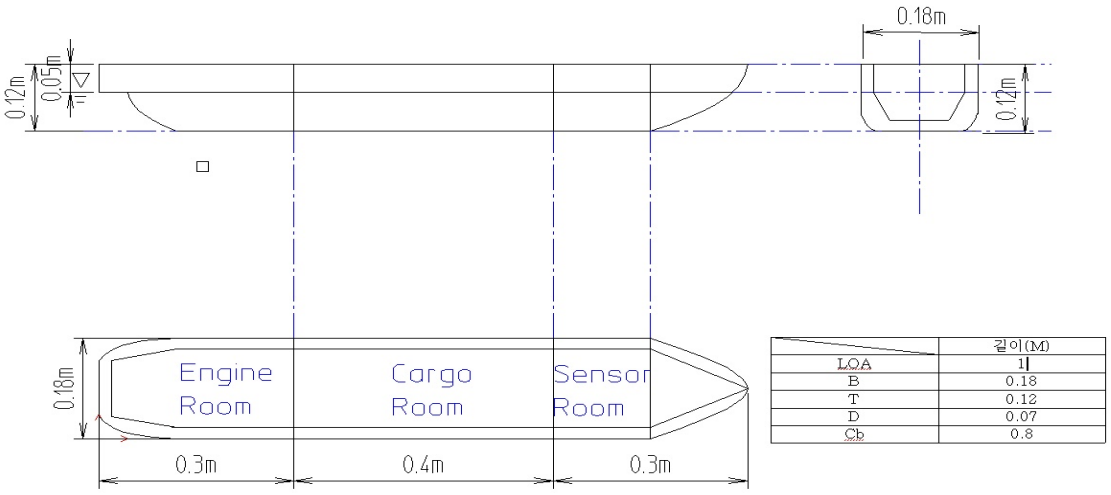
<표 2> 기관부 중량

분류		중량(kg)
추진부 및	추진축	0.05

제어부 중량	DC모터	0.05
	서브모터	0.04
	RUDDER	0.01
	수신기	0.02
	배터리	0.5
	프로펠러	0.01
	저항+ 변속기	0.06
	초음파 센서	0.10
	회로기판	0.16
계		1

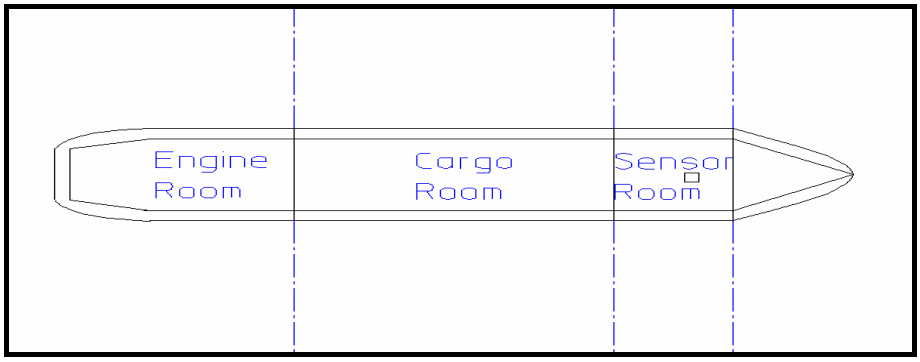
1.1.4. 개략배치도 및 구획배치도

다음은 우리 조가 제작 할 모형선박의 개략 배치도이다.



<그림 3> 모형선의 G/A

다음은 우리 조가 제작 할 모형선박의 구획 배치도이다.



<그림 4> 모형선의 구획배치도

<표 3> 구획배치

	배치될 품목
Engine Room	러더, DC모터, 서버모터, 추진축, 프로펠러, 수신기, 저항 및 변속
Cargo Room	배터리, 제어기
Sensor Room	센서,

1.2. 상세설계

1.1.1. 선형 설계 과정

본 설계를 위하여 EzShip System을 활용하는데 이는 크게 두 단계로 구분된다.

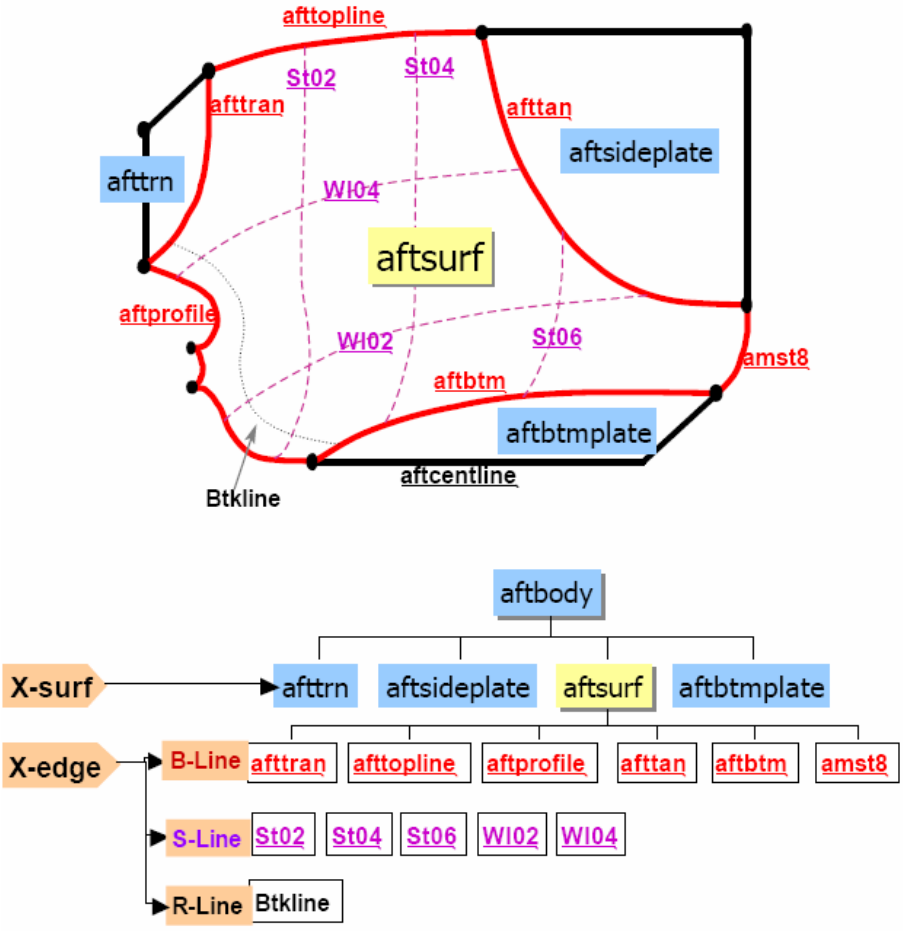
첫째 단계는 EzHULL을 이용하여 반폭 선형정의 파일(*.xaf)를 형성한다. 그 다음 이를 반폭 이전 선형 파일 (*.xan)으로 변환 후 차후 작업을 위해 Mesh 형성 작업을 거쳐 반폭 이진 Mesh 파일(*.xan)을 만들게 된다.

둘째 단계는 EzCOMPART를 활용하여 개략배치도(G/A)에 맞게 구획을 배치하고 모형선 제작에 필요한 내부 부재의 형상을 결정하게 된다. 이 과정에서는 EzHULL에서 제작 된 반폭 이진 Mesh파일을 전폭으로 확장하여 전폭 선형 이진 파일(*.xac)를 만들게 되고 이와 함께 구획 정의 파일 (*.xab)를 이용해 전폭 선형 이진 파일(*.xac)까지 형성함으로써 본 모형선 제작에 필요한 도면 출력 직전까지의 과정을 완료하게 된다.

1.2.1.1. 선박 형상 곡선의 종류 및 정의

EzHull을 이용한 선형 설계는 선박 형상을 표현하는 그물망(line)을 형성하는 것으로부터 시작되는데 그에 필요한 Line의 종류에 대한 설명은 다음과 같다.

- Bline : Boundary line(경계선) – 다른 선에 영향을 받지 않는다. Xsurf를 구성할 때 가장 외곽은 bline이어야 한다.
- Sline : Surface line(교차선) – 경계선이나, 다른 교차 선에 영향을 받는다. 곡면 패치를 구성하는 단위.
- Aline : 공간선 정의를 위한 보조선을 정의할 때 사용



<그림 5 > Line 의 종류[1]

1.2.1.2. 선형 정의 파일의 제작

앞서 소개한 line들을 이용하여 선형 표현을 위한 주요 곡선을 나타내고 선형 곡면 모델을 생성하기 위한 선형 정보 파일인 xaf 파일을 다음과 같이 제작하였다.

```
$PROJ = "renovatio";
```

[1] asdal 연구실 제작 ezHull 매뉴얼

```
$DATE = "2006.9.29";
$USER = "renovatio";

unit 1.000000;
xframe (1.0000000000);
yframe (1.0000000000);
zframe (1.0000000000);

$L = 0.95000000;
interval $L/20;
$B = 0.18000000;
$D = 0.12000000;
$T = 0.07000000;
$TA = 0.07000000;
$TF = 0.07000000;
$LOA = 1.00000000;
$PSB = 0.28500000;
$PSE = 0.66500000;
$SCH = 0.00000000;
$PRODIA = 0.00000000;
$BULBLEN = 0.00000000;
$SPEED = 0.00000000;
$BILGE = 1.00000000;
$RBILGE = 0.01000000;
$RBILGE2 = 0.00000000;
$KEEL = 0.00000000;
$DEADRISE = 0.00000000;

form FORM_sa1 {

        point tranedge (-0.03000000, 0.00000000, 0.06000000);
        point tranctop (-0.03000000, 0.00000000, 0.12000000);
        point transtop (-0.03000000, 0.04500000, 0.12000000);
        point atep (0.16000000, 0.09000000, 0.12000000);
        point abep (0.16000000, 0.00000000, 0.00000000);
        point amcp (0.28500000, 0.00000000, 0.00000000);
```

```
point amsp (0.28500000, 0.08650000, 0.00000000);
point abilgetop (0.28500000, 0.09000000, 0.00350000);
point amtp (0.28500000, 0.09000000, 0.12000000);


bline   st_tran   X=-0.03000000    /K   {
          transtop /K /WP;
          <0.04129253,   0.08907600>;
          <0.03875582,   0.07682049>;
          tranedge /K /WPE;
}

bline   amst8.8   X=0.28500000    /K   {
          amcp /WP;
          <0.02883333,   0.00000000>;
          <0.05766667,   0.00000000>;
          amsp /T;
          (0.09000000,   0.00000000) /R;
          abilgetop /T;
          <0.09000000,   0.04233333>;
          <0.09000000,   0.08116667>;
          amtp /WPE;
}

bline   aftbtm   Z=0.00000000    /T   {
          abep /K /WP;
          <0.16993334,   0.00592575>;
          <0.18698585,   0.03267190>;
          (0.21018701,   0.05033980) /K;
          <0.23730826,   0.07099289>;
          <0.27283146,   0.07924083>;
          amsp /WPE;
}

bline   afttopline   Z=0.12000000    /K   {
          tranctop /WP;
```

```
            <-0.02999097,   0.01499934>;
            <-0.03000000,   0.03000000>;
            transtop /K;
            <-0.02982890,   0.04504529>;
            <0.00808932,   0.06548894>;
            (0.04847445,   0.07664222) /K;
            <0.09262134,   0.08883440>;
            <0.13971613,   0.08992486>;
            atep /K;
            <0.17261753,   0.09000000>;
            <0.25238247,   0.09000000>;
            amtp /WPE;
        }

        bline   aftcentline   Y=0.00000000    /K   {
            abep /WP;
            <0.20166667,   0.00000000>;
            <0.24333333,   0.00000000>;
            amcp /WPE;
        }

        bline   aftpro   Y=0.00000000    /K /P   {
            tranctop /WP;
            <-0.03000000,   0.10000000>;
            <-0.03000000,   0.08000000>;
            tranedge /K;
            <0.10294656,   0.06003522>;
            <0.12590590,   0.00000000>;
            abep /WPE;
        }

        bline   afttan   Y=0.09000000    /T   {
            atep /WP;
            <0.15886468,   0.06947133>;
            <0.17123184,   0.04103847>;
            abilgetop /K /WPE;
```

```
        }

sline   st_1   X=0.06    {
                           (@aftpro);
                           (@afttopline);
                }
sline   st_2   X=0.12    {
                           (@aftpro);
                           (@afttopline);
                }
sline   st_3   X=0.18    {
                           (@aftcentline);
                           (@aftbtm) /K;
                           (@afttopline);
                }
sline   st_4   X=0.24    {
                           (@aftcentline);
                           (@aftbtm) /K;
                           (@afttopline);
                }

        xsurf sa1surf { "*" }


}

form FORM_pa1 {



        bline   pmst8.8   X=0.28500000    /K   {
                (0.00000000,   0.00000000) /WP;
                <0.02883333,   0.00000000>;
                <0.05766667,   0.00000000>;
                (0.08650000,   0.00000000) /T;
                (0.09000000,   0.00000000) /R;
```

```
            (0.09000000,   0.00350000) /K;
            <0.09000000,   0.04233333>;
            <0.09000000,   0.08116667>;
            (0.09000000,   0.12000000) /WPE;
    }

    bline   pmst15   X=0.66500000    /K   {
            (0.00000000,   0.00000000) /WP;
            <0.02883333,   0.00000000>;
            <0.05766667,   0.00000000>;
            (0.08650000,   0.00000000) /T;
            (0.09000000,   0.00000000) /R;
            (0.09000000,   0.00350000) /K;
            <0.09000000,   0.04233333>;
            <0.09000000,   0.08116667>;
            (0.09000000,   0.12000000) /WPE;
    }

    bline   parbtm   Z=0.00000000    /T   {
            (0.28500000,   0.08650000) /WP;
            <0.41166667,   0.08650000>;
            <0.53833333,   0.08650000>;
            (0.66500000,   0.08650000) /WPE;
    }

    bline   partopline   Z=0.12000000     {
            (0.28500000,   0.09000000) /WP;
            <0.41166667,   0.09000000>;
            <0.53833333,   0.09000000>;
            (0.66500000,   0.09000000) /WPE;
    }

    bline   parcentline   Y=0.00000000    /K   {
            (0.28500000,   0.00000000) /WP;
            <0.41166667,   0.00000000>;
            <0.53833333,   0.00000000>;
```

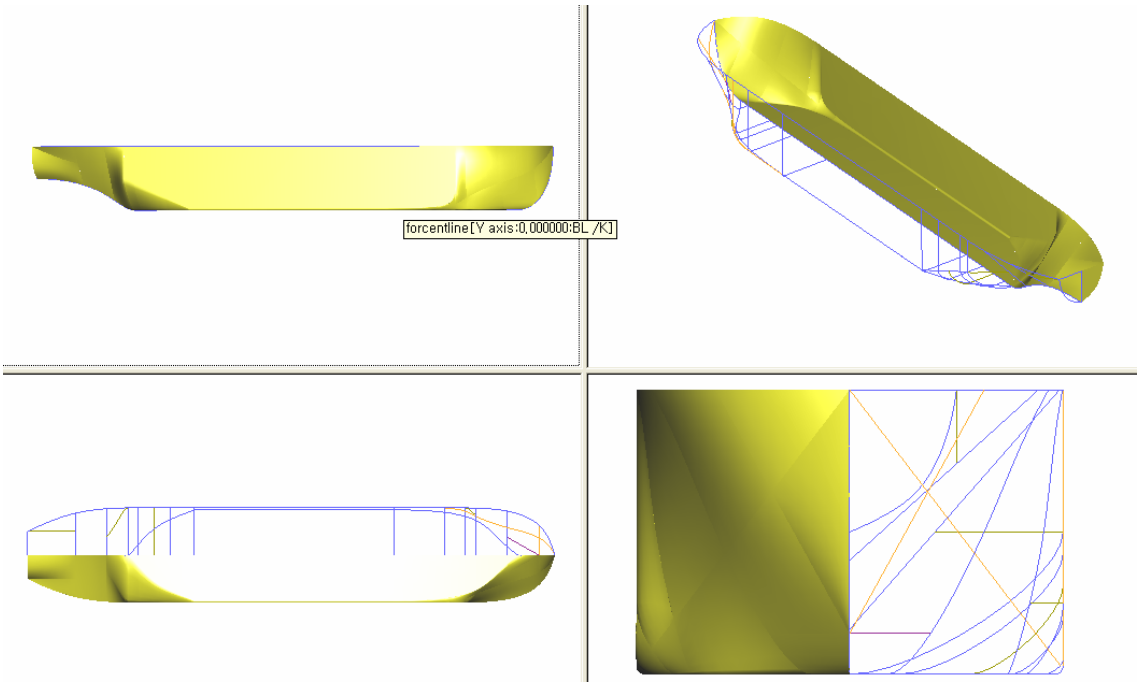
```
                (0.66500000,    0.00000000) /WPE;
        }

        bline   bilgetopline   Y=0.09000000    /T   {
                (0.28500000,    0.00350000) /WP;
                <0.41166667,    0.00350000>;
                <0.53833333,    0.00350000>;
                (0.66500000,    0.00350000) /WPE;
        }



        xsurf parrsurf {
                pmst8.8 pmst15 parbtm partopline parcentline bilgetopline
        }

}

form FORM_sf1 {

        point fmcp (0.66500000, 0.00000000, 0.00000000);
        point fmsp (0.66500000, 0.08650000, 0.00000000);
        point fbilgetop (0.66500000, 0.09000000, 0.00350000);
        point fmtp (0.66500000, 0.09000000, 0.12000000);
        point ftep (0.80000000, 0.09000000, 0.12000000);
        point fbep (0.90000000, 0.00000000, 0.00000000);
        point fmosttop (0.97000000, 0.00000000, 0.12000000);


        bline   fmst15   X=0.66500000    /K   {
                fmcp /WP;
                <0.02883333,    0.00000000>;
                <0.05766667,    0.00000000>;
                fmsp /T;
                (0.09000000,    0.00000000) /R;
                fbilgetop /K;
```

```
            <0.09000000,   0.04233333>;
            <0.09000000,   0.08116667>;
            fmtp /WPE;
        }

        bline   forbtm   Z=0.00000000    /T   {
            fmsp /WP;
            <0.85736722,   0.08572635>;
            <0.82822603,   0.05094805>;
            fbep /WPE;
        }

        bline   fortopline   Z=0.12000000    /K   {
            ftep /WP;
            <0.92303318,   0.08994621>;
            <0.95299172,   0.06156215>;
            fmosttop /WPE;
        }

        bline   fortopline_2   Z=0.12000000    /K   {
            fmtp /WP;
            <0.71000000,   0.09000000>;
            <0.75500000,   0.09000000>;
            ftep /WPE;
        }

        bline   forcentline   Y=0.00000000    /K   {
            fmcp /WP;
            <0.74333333,   0.00000000>;
            <0.82166667,   0.00000000>;
            fbep /WPE;
        }

        bline   forpro   Y=0.00000000    /K   {
            fbep /WP;
            <0.93998367,   0.00000000>;
```

```
                <0.97000000,   0.04000000>;
                fmosttop /WPE;
        }

        bline   fortan   Y=0.09000000    /T   {
                fbilgetop /K /WP;
                <0.78365338,   0.00344489>;
                <0.79563612,   0.00592933>;
                (0.80000000,   0.03000000) /K;
                <0.80000000,   0.06000000>;
                <0.80000000,   0.09000000>;
                ftep /K /WPE;
        }
sline   st_10   X=0.76:     {
                (@forcentline);
                (@forbtm) /K;
                (@fortan) /K;
                (@fortopline_2);
        }
sline   st_11   X=0.82:     {
                (@forcentline);
                (@forbtm) /K;
                (@fortopline);
        }
sline   st_12   X=0.88:     {
                (@forcentline);
                (@forbtm) /K;
                (@fortopline);
        }
sline   st_13   X=0.94:     {
                (@forpro);
                (@fortopline);
        }
        xsurf forsurf { "*" }

}
```

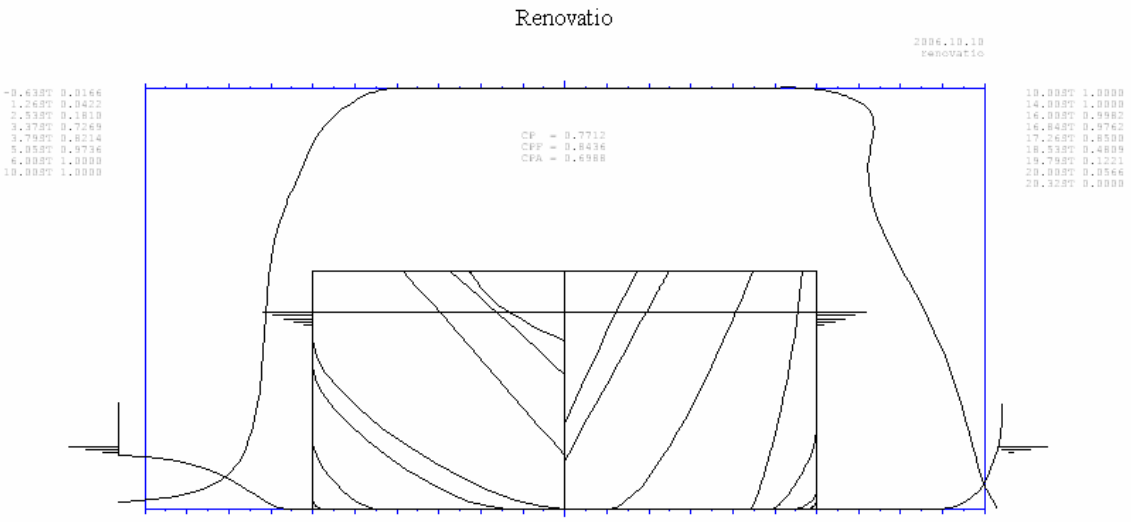
1.2.1.3 선형 곡면 모델 생성 및 곡면 fairing

이렇게 일차적으로 작성된 xaf파일을 이용하여 선형 곡면 모델을 생성해 본 결과 다음과 같았다.



<그림 6> fairing 전 선형 곡면 모델

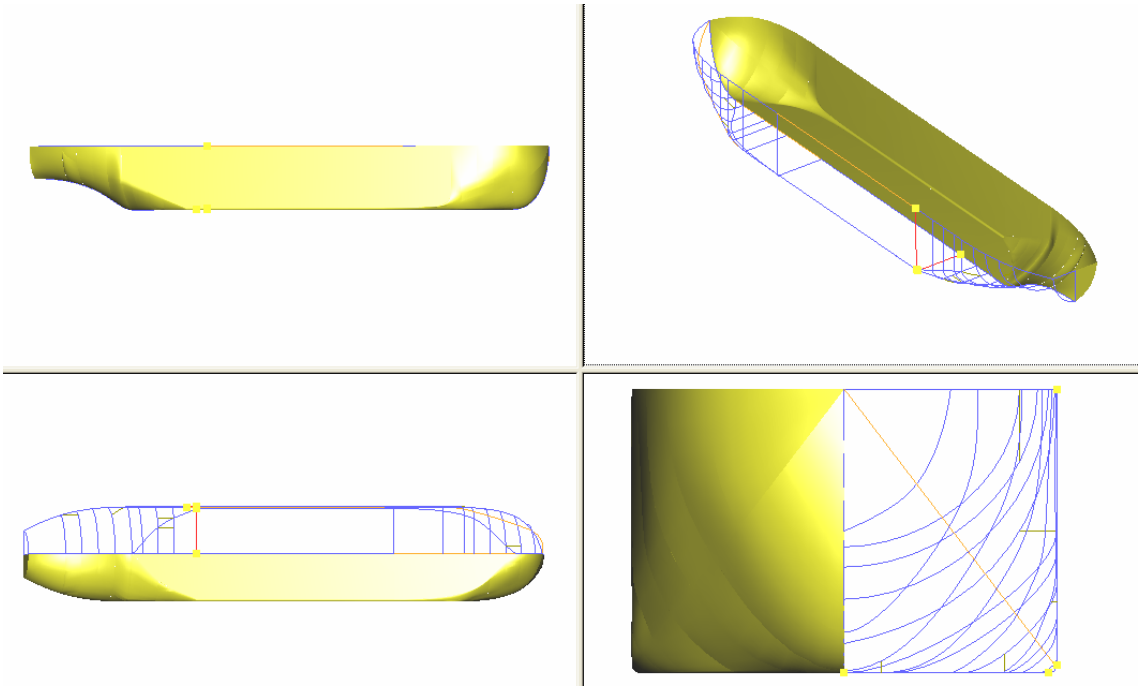
육안으로도 선수부의 곡면이 매끄럽지 못하고 섹션 간에 선명하게 명암이 갈린 모습을 볼 수 있다. 이는 곡면 fairing의 필요성을 의미한다. 이를 자세하기 확인하기 위하여 cp curve를 출력해보았다.



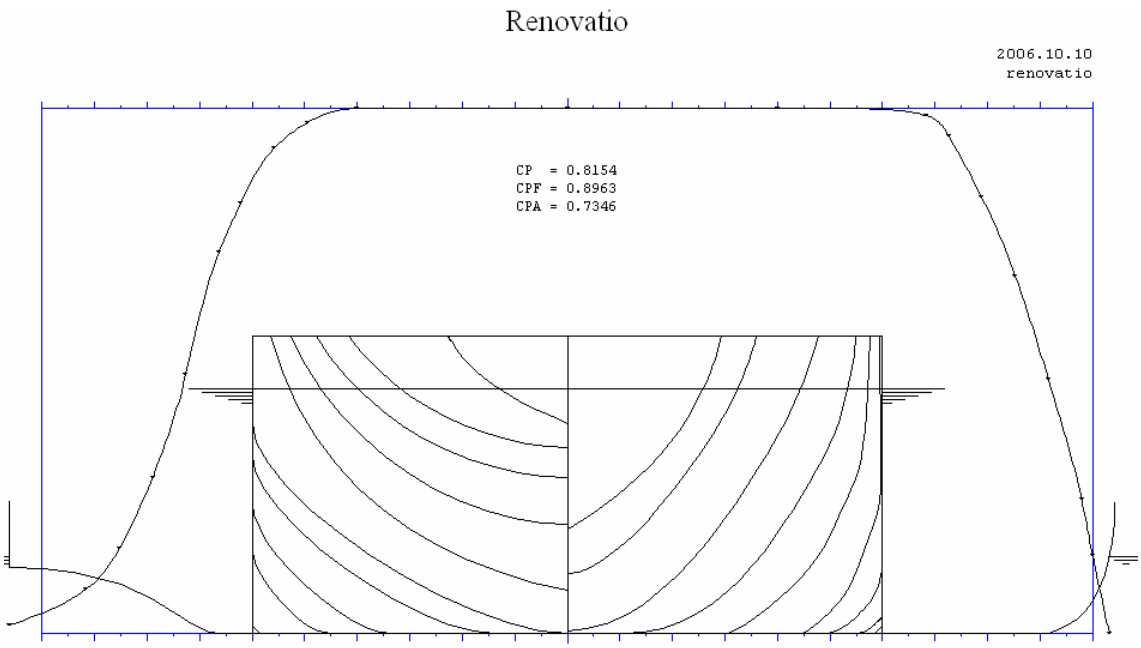
<그림 7>fairing 전 cp curve

출력된 cp curve에서도 선수부의 곡선이 매끄럽지 못함을 알 수 있다. 이것은 횡단면적의 불규칙적인 변화를 의미하며 선체 저항에 영향을 미쳐 배의 성능에 악영향을 줄 수 있다.

따라서 선수부와 선미부의 side라인을 더 추가하고 직선을 곡선으로 보간해 준 후 cp curve를 확인하는 과정을 여러 번 거쳐 마침내 바람직한 선형을 얻게 되었다. Fairing 후 선형 곡면 모델 및 cp curve는 다음과 같다.



<그림 8 >fairing 후 선형 곡면 모델



<그림 9> fairing 후 cp curve

```
L O A = 100.00 M    L/B     =  5.2778 LCB =     3.94 % LPP
L B P =  95.00 M    B/T     =  2.5714 VCB =     3.68 M
BEAM  =  18.00 M    CB      =  0.8151 KMT =   7.6797 M
DEPTH =  12.00 M    CBF     =  0.8959 LE  =   28.500 M(30.00 %)
DRAFT =   7.00 M(TF)CBA     =  0.7343 LX  =   38.000 M(40.00 %)
      =   7.00 M(TA)CM      =  0.9996 LR  =   28.500 M(30.00 %)
SPEED =   0.00 KTS  CW      =  0.9363 AFP =   18.796 M^2(14.92 % AM)
FN    =   0.0000    CWF     =  0.9584 BL  =    0.000 M( 0.00 % LPP)
SCH   =   0.000 M   CWA     =  0.9142 WSA =   2607.0 M^2
DIA   =   0.000 M   CB/(L/B)=  0.1544 VOL =   9756.6 M^3
```

<그림 10> fairing 후 cp curve 상세 정보

fairing 후 cp curve를 보게 되면 fairing 전에 비해 선수부 선미부 모두 부드러워 진 것을 볼 수 있다. 일차적으로 설계 완료된 모형선의 주요 치수는 다음과 같다.

```
L O A = 100.00 M
L B P =  95.00 M
BEAM  =  18.00 M
DEPTH =  12.00 M
DRAFT =   7.00 M(TF)
      =   7.00 M(TA)
```

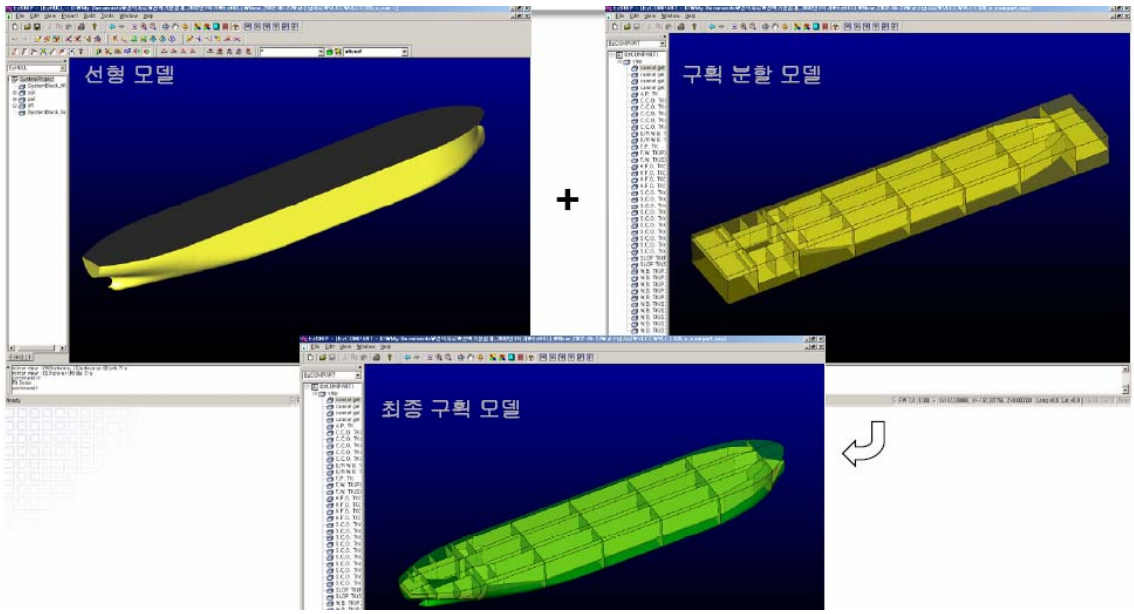
<그림 11> 일차 선형 주요치수

(Cp 커브에서 정보가 M단위로 나오는데 이는 xaf 스트립트 수정 시 좌표를 m단위로 입력한 탓이다)

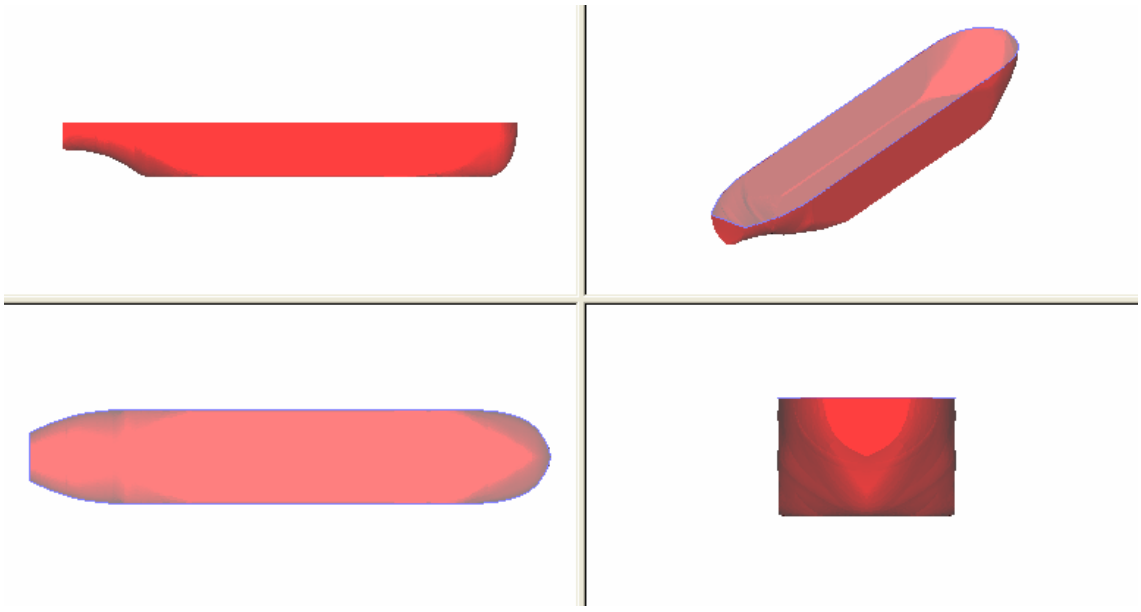
위 cp curve 상세 정보를 보게 되면 방형계수(Cb)는 0.8154 만재 배수량은 9.756kg 으로 Proposal 당시 결정된 주요치수와 거의 동일하게 일차 선형 설계가 완료되었다.

1.2.2. 구획 설계 과정

EzCompart 를 이용하여 구획 모델링을 하는데 앞서 제작된 선형 모델과 구획 분할 모델을 합쳐 최종 구획 모델을 형성하게 된다.



<그림 12 >최종 구획 모델 생성 과정



<그림 13> 전폭 선형 이진 파일의 모습

위의 진폭 선형 이진 파일과 합성할 구획 분할 모델을 스크립트로 작성하게 되는데 작성한 스크립트는 다음과 같다.

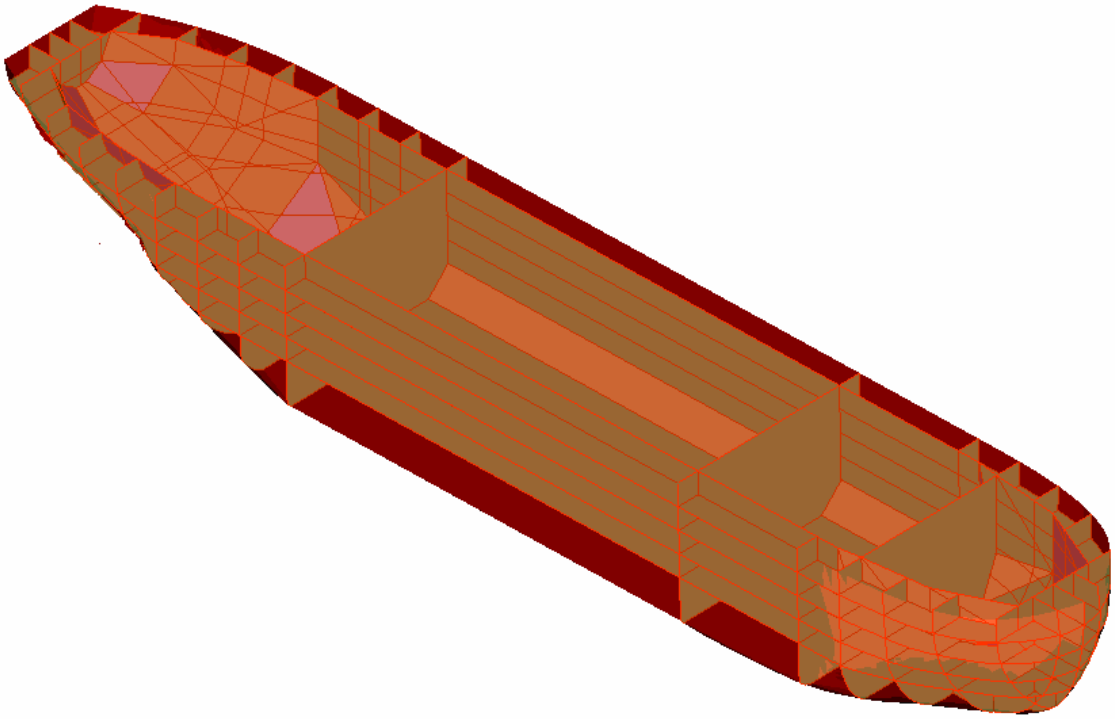
```
load hull /d "union.xac";

//variable

//box ship

box ship /e { (-3, -9, 0), (97, 9, 12)};
base ship {
//격벽
        xplane end 28.5;
        place (end) @{*}
        xplane for 66.5;
        place (for) @{*}
        xplane coll 82;
        place (coll) @{*}
        xplane coll1 94;
        place (coll1) @{*}
        xplane end1 0;
        place (end1) @{*}

polyline af1 x=0 { (-3, 12), (-2, 10), (-0.5, 9), (0.5, 9), (2, 10), (3, 12) };
polyline af2 x=4 { (-5.4, 12), (-5, 10), (-2, 7.5), (2, 7.5), (5, 10), (5.4, 12) };
polyline af3 x=10 { (-6.5, 12), (-5, 7.5), (-2.3, 5.5), (2.3, 5.5), (5, 7.5), (6.5, 12) };
polyline af4 x=16 { (-7, 12), (-7, 6.5), (-0.1, 2), (0.1, 2), (7, 6.5), (7, 12) };
polyline af5 x=21 { (-7, 12), (-7, 4), (-4.7, 2), (4.7, 2), (7, 4), (7, 12) };
polyline af6 x=24 { (-7, 12), (-7, 4), (-5, 2), (5, 2), (7, 4), (7, 12) };
polyline af7 x=28.5 { (-7, 12), (-7, 4), (-5, 2), (5, 2), (7, 4), (7, 12) };

polyline deck2 x=66.5 { (-7, 12), (-7, 4), (-5, 2), (5, 2), (7, 4), (7, 12) };
```

```
polyline sensor2 x=80 { (-7, 12), (-7, 4), (-5, 2), (5, 2), (7, 4), (7, 12) };
polyline sensor3 x=82 { (-6.7, 12), (-6.5, 5.5), (-5, 2), (5, 2), (6.5, 5.5), (6.7, 12) };
polyline fo2 x=88 { (-6.1, 12), (-6, 7), (-2.5, 2.2), (2.5, 2.2), (6, 7), (6.1, 12) };
polyline fo3 x=94 { (-3.5, 12), (-3.5, 9), (-0.1, 4), (0.1, 4), (3.5, 9), (3.5, 12) };
skinsurf compart /e {af1, af2, af3, af4, af5, af6, af7, deck2,   sensor2, sensor3, fo2,fo3};
place (compart) @{*}

        zplane z1 4;
        place (z1) @{+compart,-end1, +coll1}
        zplane z2 6;
        place (z2) @{+compart,-end1, +coll1}
        zplane z3 8;
        place (z3) @{+compart,-end1, +coll1}
        zplane z4 10;
        place (z4) @{+compart,-end1, +coll1}

        xplane x1 78;
        place (x1) @{+compart}
        xplane x3 86;
        place (x3) @{+compart}
        xplane x4 90;
        place (x4) @{+compart}
        xplane x5 94;
        place (x5) @{+compart}
        xplane f1 0;
        place (f1) @{+compart}
        xplane f2 4;
        place (f2) @{+compart}
        xplane f3 8;
        place (f3) @{+compart}
        xplane f4 12;
        place (f4) @{+compart}
        xplane f5 16;
```

```
        place (f5) @{+ compart}
        xplane f6 20;
        place (f6) @{+ compart}
        xplane f7 24;
        place (f7) @{+ compart}

        yplane y1 2;
        place (y1) @{+ compart, -end1, + coll1}
        yplane y2 -2;
        place (y2) @{+ compart,-end1, + coll1}
        yplane cen 0;
        place (cen) @{+ compart,-end1, + coll1}

        place (hull) @{*};

}
```

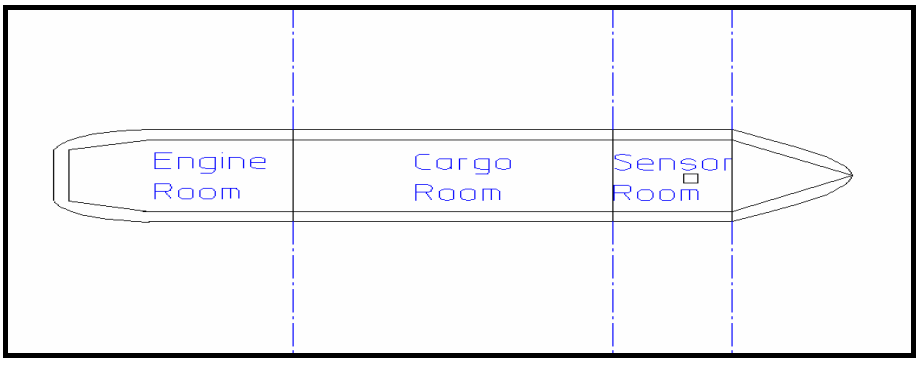
위의 스크립트는 격벽 배치를 통한 구획 분할 뿐만 아니라 모형선 제작 시 필요한 부재의 모양을 결정하는 내부 블록의 모양을 설정한다. 위의 스크립트 중 polyline이 내부 부재의 안쪽 형상을 결정하는 라인을 형성하게 되고 U자 횡방향 부재의 안쪽 굴곡은 이 polyline에 의해 결정되게 된다.

Xplane은 격벽 및 횡방향 부재의 위치를, yplane은 용골의 위치를, zplane은 종방향(테두리) 부재위 위치를 결정하게 된다. 아래의 그림은 물론 실제 부재의 모양은 아니다. 실재 네스팅 및 커팅하게 될 부재의 모양은 drawGA를 통해 각 부재의 절단면을 AUTOCAD파일로 export함으로써 도면으로 출력된다.



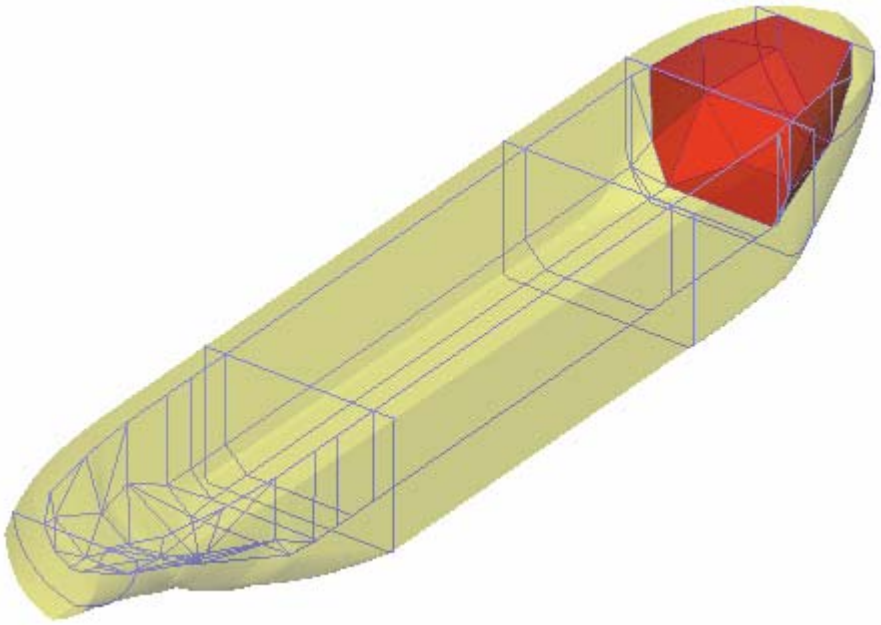
<그림 14> 선형 이진 파일과 구획 분할 모델을 합성

<그림 9>과 <그림 10>을 참조하면 Proposal 시와 동일하게 격벽을 배치하여 구획을 나눠놓은 것을 알 수 있다.

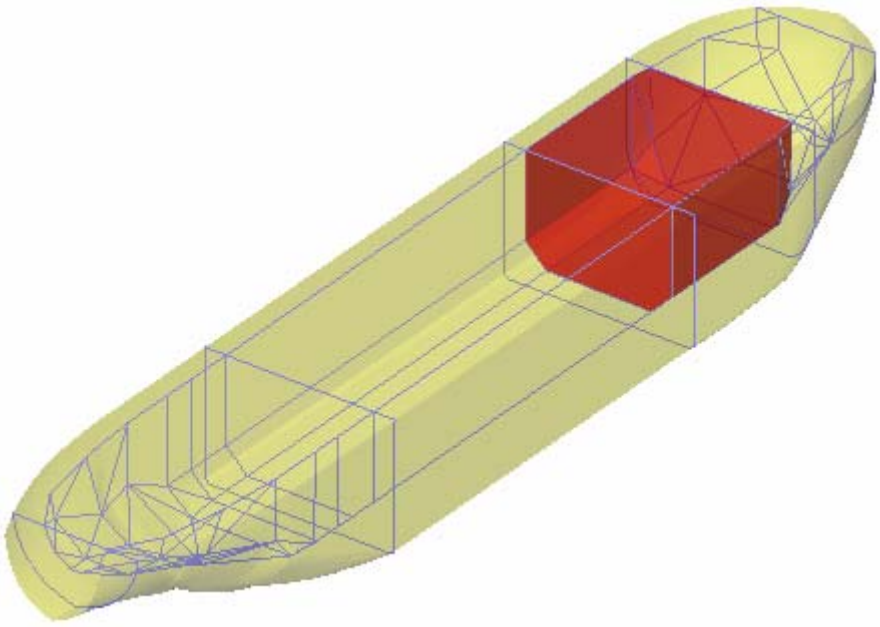


<그림 15> Proposal 당시의 구획 배치도

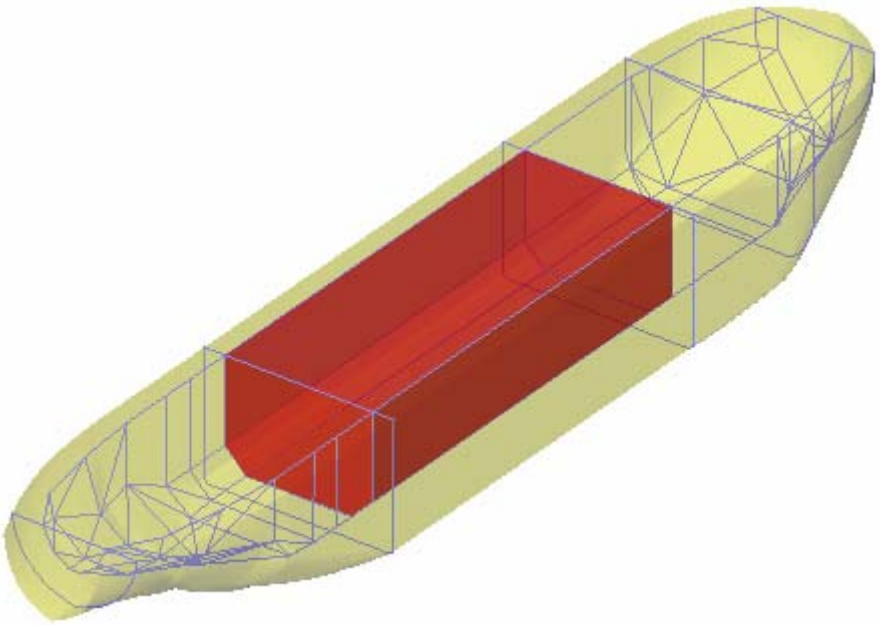
아래 그림들에서 빨간 색으로 반전된 공간이 하나의 구획이다.



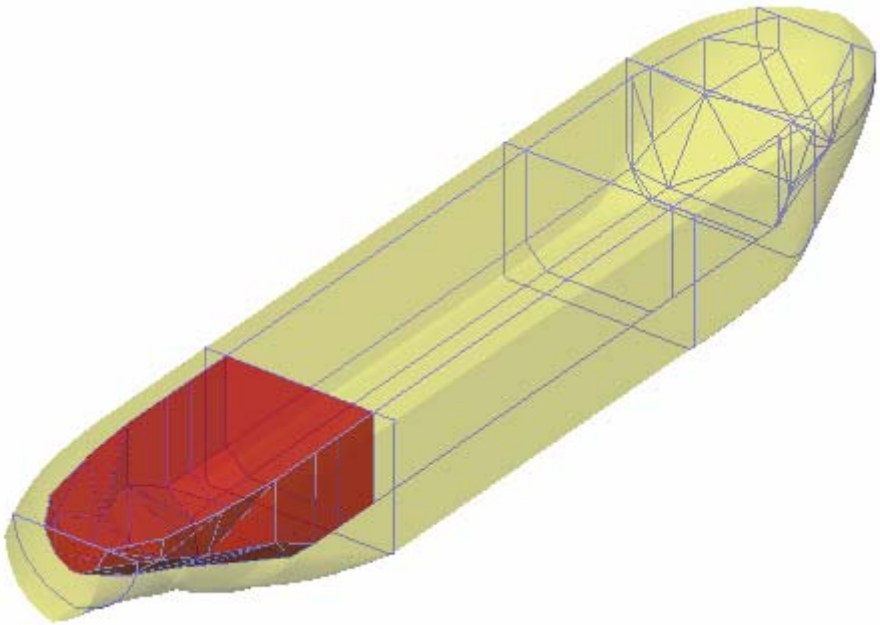
<그림 16> 선수부



<그림 17> Sensor Room



<그림 18> Cargo Room



<그림 19> Engine Room

<표 4> 각 구획별 배치 품목

	배치될 품목
Engine Room	러더, DC모터, 서버모터, 추진축, 프로펠러, 수신기, 저항 및 변속
Cargo Room	배터리, 제어기
Sensor Room	센서,

1.2.3. 세부 경하중량의 추정

EzHull과 EzCompart 를 이용하여 선박 형상 및 구획 배치를 끝내게 되면 실제 모형선 제작 시 조립하게 될 부재의 도면까지 출력할 수 있게 된다. 따라서 이 단계에서는 Proposal시의 1차 개략 경하중량 추정보다 더 정확하게 세부경하 중량 추정이 가능하다. 각 부재의 단면적과 선각 면적을 EzHull 및 AutoCAD 의 도움을 받아 정확하게 계산할 수 있고 이를 통해 각 부재 및 선각의 부피를 구하여 그 밀도를 곱해 계산해주면 선체중량을 구할 수 있게 된다. 이에 의장에 해당하는 기관부 중량을 더 해주면 우리 조가 앞으로 제작하게 될 모형선의 세부 경하중량 추정이 가능하게 된다.

1.2.3.1.선체중량

선체 중량을 Hull에 해당하는 선각 중량과 내부 부재로 나누어 생각할 수 있다.

선각 중량

선각 중량은 선각 부분의 총 단면적에 그 재료의 두께와 밀도를 곱하여 그 중량을 추정하게 된다. 이 때 단면적의 정확한 계산이 요구되는데 이는 EzHull 프로그램을 통해서 쉽게 구할 수 있다. 한편, 선각의 단면적은 결국 배 전체가 물에 잠겼을 때 침수 면적과 같다. 따라서 EzHull 툴을 이용해 흘수를 배의 Depth 까지 올려서 침수 면적을 구함으로써 선각의 단면적을 알 수 있게 된다. 이는 cp curve에서 확인할 수 있는데 흘수를 본 조의 모형선의 Depth 인 12까지 올려서 cp curve를 통해 단면적을 알아본 결과 다음과 같았다.

- 선각의 단면적 : 3694.9 cm^2

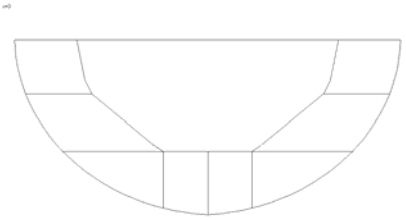
이를 이용해 선각 부분을 구성하게 되는 우드락, 에폭시 및 퍼티 각각 의 무게 및 선각 총 중량을 표로 정리해보면 다음과 같다.

<표 5> 선각 중량

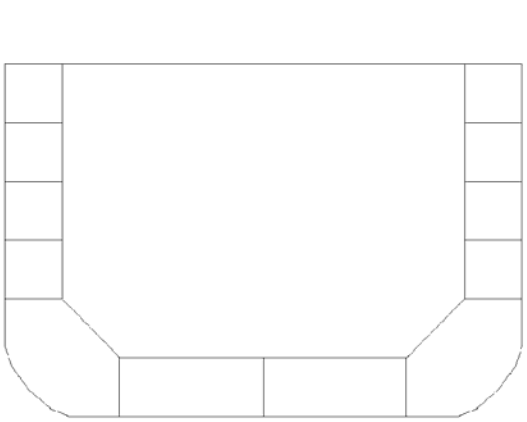
	두께(cm)	부피(cm^3)	중량(g)
우드락	0.2	738.98	517.286
에폭시	0.05	184.745	277.1175
퍼티	0.1	369.49	498.8115
계			1293.215

1.2.3.2. 내부 부재 중량

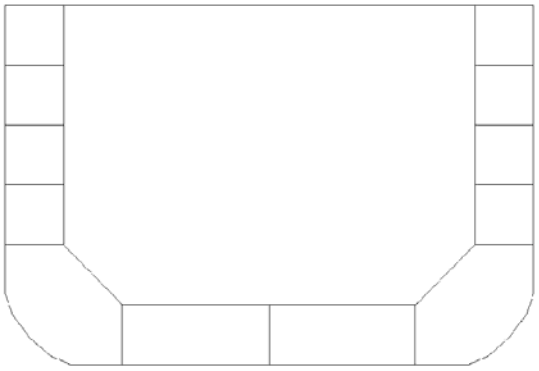
선각 부분과 마찬가지로 내부 부재 중량도 그 면적을 계산한 후 부피를 구하는 고정으로 추정하는데 앞서 설명한 바와 같이 AutoCAD 프로그램의 Area 명령어를 이용함으로써 내부 부재들의 넓이를 정확하고 간단하게 계산할 수 있다. AutoCAD에서 화면 출력해본 내부 부재의 모양들은 다음과 같았다. 위치는 모두 cm 단위이다.



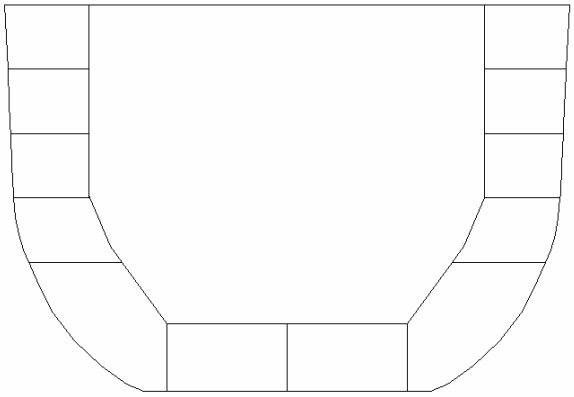
<그림 20> x=3에서의 단면



<그림 21> x=24에서의 단면

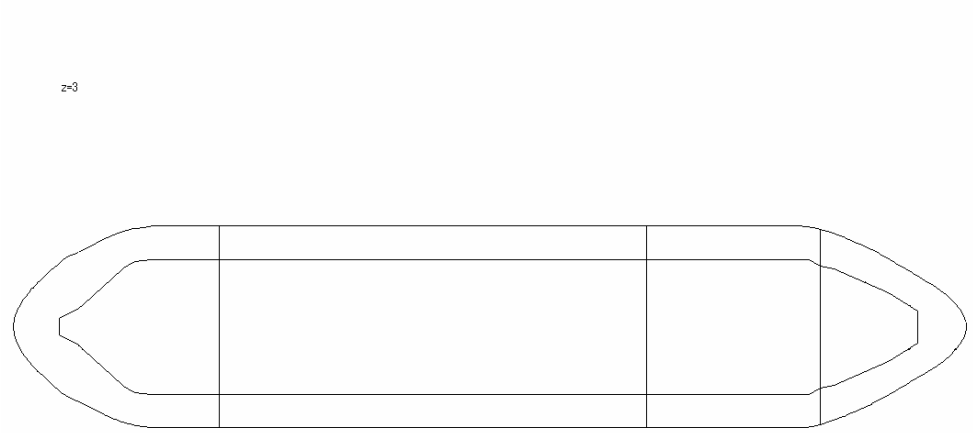


<그림 22> 중앙평행부 단면

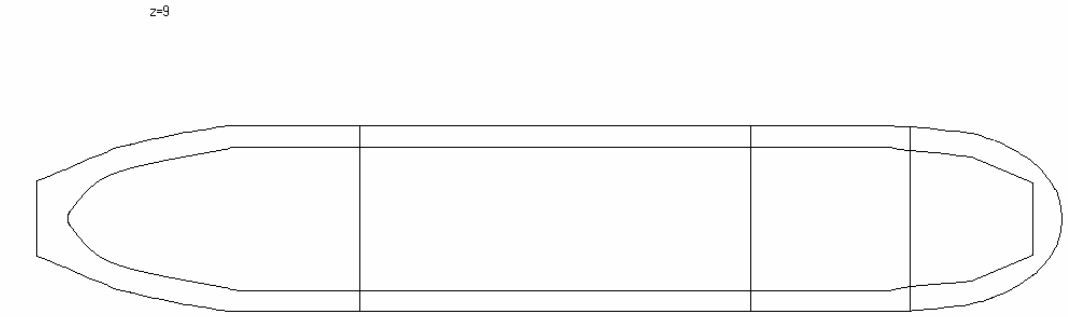


<그림 23> x=86에서의 단면

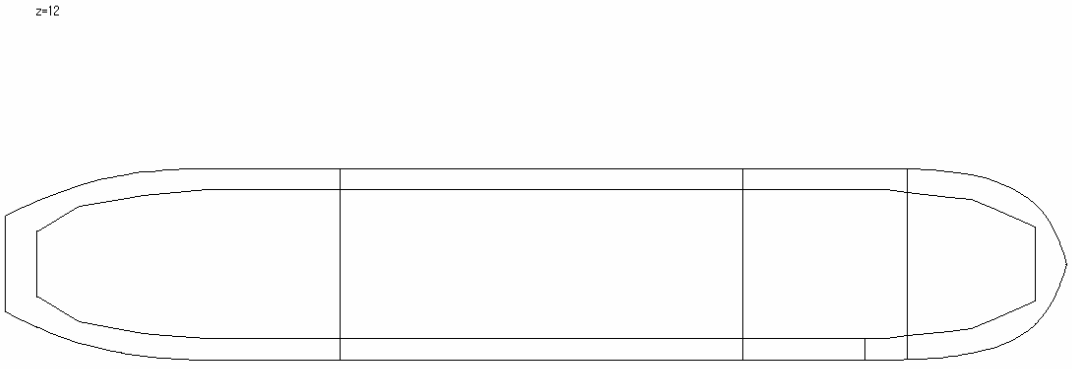
<그림 24> y=0 에서의 단면



<그림 25> z=3에서의 단면



<그림 26> z=9에서의 단면



<그림 27> z=12에서의 단면

이러한 단면들을 AutoCAD를 통해 각 x, y, z 위치에서의 단면적, 즉 부재의 면적을 구하고 그 중량들을 계산해 본 결과를 표로 정리해 보았다. 내부 부재들은 두께는 3mm이다.

<표 6> 횡방향 U자 부재들의 각 면적 및 중량

위치(x= cm)	단면적(cm^2)	부피(cm^3)	중량(g)	비고
3	39.14200042	11.74260013	8.219820089	
6	35.30009734	10.5900292	7.413020441	
9	46.26568126	13.87970438	9.715793064	
12	59.06318814	17.71895644	12.40326951	
15	71.59668849	21.47900655	15.03530458	
18	63.56424639	19.06927392	13.34849174	
21	57.99999941	17.39999982	12.17999988	
24	57.99999941	17.39999982	12.17999988	
27	57.99999941	17.39999982	12.17999988	
28.5	215.8774977	64.76324931	45.33427452	격벽
중앙평행부	753.9999923	226.1999977	158.3399984	13 개
68.5	215.8774977	64.76324931	45.33427452	격벽
70	57.99999941	17.39999982	12.17999988	
73	57.99999941	17.39999982	12.17999988	
76	57.99999941	17.39999982	12.17999988	
79	57.99999941	17.39999982	12.17999988	
82	205.3313744	61.59941231	43.11958862	격벽
85	65.35257854	19.60577356	13.72404149	
88	62.09928942	18.62978683	13.04085078	
91	64.3300044	19.29900132	13.50930092	

94	81.37771691	24.41331507	17.08932055	
계			500.8873483	

<표 7> 용골 부재 면적 및 중량

위치(y= cm)	단면적(cm^2)	부피(cm^3)	중량(g)	비고
0	234.7416908	70.42250724	49.2958	
5	232.780807	69.8342421	48.884	
-5	232.780807	69.8342421	48.884	
계			147.064	

<표 8> 종방향 부재 면적 및 중량

위치(z= cm)	단면적(cm^2)	부피(cm^3)	중량(g)	비고
3	649.7892057	194.9367617	136.456	
6	590.5470006	177.1641002	124.015	
9	466.6622855	139.9986856	97.9991	
12	449.5781977	134.8734593	94.4114	
계			452.881	

위의 단면 및 위치별 부재 중량을 종합하여 내부 부재의 총 중량을 추산해보면 다음과 같다.

<표 9> 내부 부재별 중량 합계

	두께(cm^2)	부피(cm^3)	중량(g)
횡방향부재	0.2	738.98	500.8873483
종방향부재	0.05	184.745	452.881
용골	0.05	184.745	98.0425
계			1100.832

1.2.3.3. 기관부 중량

기관부 중량은 앞서 계산한 선체부 중량에 비해 고정적이고 실측 중량으로 그 무게를 쉽게 알 수 있다 그 내역과 중량은 다음과 같다.

<표 10> 기관부 중량 내역

분류		중량(kg)
추진부 및 제어부 중량	추진축	0.05
	DC모터	0.05
	서브모터	0.04
	RUDDER	0.01
	수신기	0.02
	배터리	0.5
	프로펠러	0.01
	저항+ 변속기	0.06

	초음파 센서	0.10
	회로기판	0.16
계		1

1.2.3.4. 총 경하중량의 추정과 재화중량

앞서 부분별로 구한 중량을 합산하여 표로 나타내면 다음과 같다.

<표 11> 총 경하중량

분류		중량(g)	계(g)
선체중량	선각부	1293.215	2394.047147
	내부		
	부재	1100.832147	
기관부중량		1000	1000
계			3394.047147

이를 통해 본 설계선의 재화중량을 계산해 보았다.

W=LWT+ DWT

에서

만재배수량 W=9.756kg 이었고 [2] 경하중량 LWT=3.494kg으로 추정되었으므로 본 설계선은 DWT=W-LWT 에서 6.262kg의 재화중량을 가질 것으로 추정된다. 이는 요구조건인 재화중량 6kg을 약간 상회하는

[2] 1.1절 참조

것으로써 약간의 마진을 고려해야하므로 설계선의 추가적인 흘수나 선형 변형이 필요 없다는 것을 알 수 있었다. 이는 Proposal 당시의 개략 경하중량 추정값[3]과 이번 EzShip System과 AutoCAD를 이용한 세부 경하 중량 추정 값이 거의 일치 했기 때문이라고 판단된다. 이로써 1차 설계 그대로 모형선 제작에 착수하게 되었다.

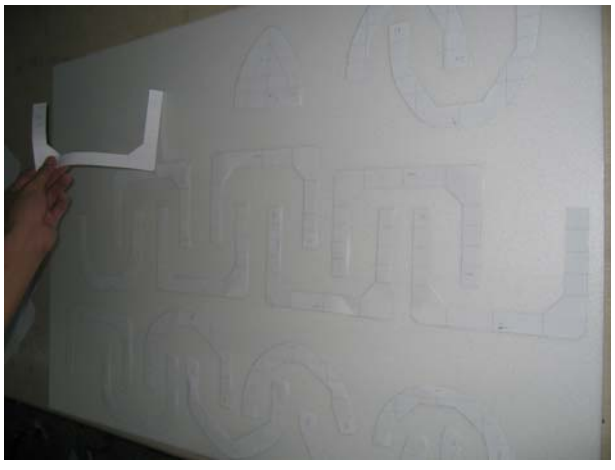
2. 선체부 제작

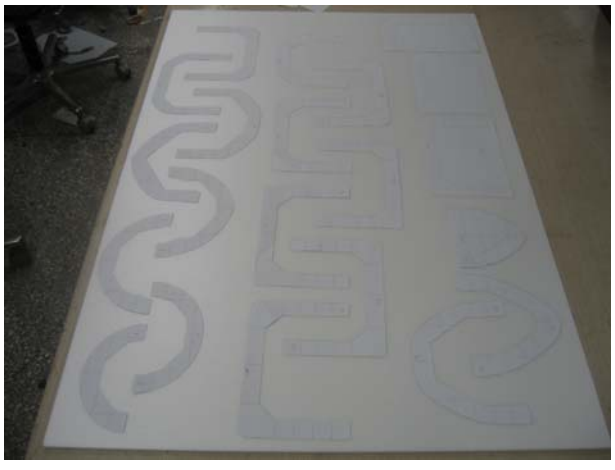
Nesting and Cutting 작업.

Ez compart로 구획설계까지 마친 후 .dxf파일로 export 한 후 AutoCad를 이용해 도면을 출력하고 출력된 도면 부재를 우드락에 붙여 Nesting & Cutting 작업을 한다.

2.1.Nesting

부재 소모를 줄이기 위해 최적의 배치를 하는 작업을 하였다.



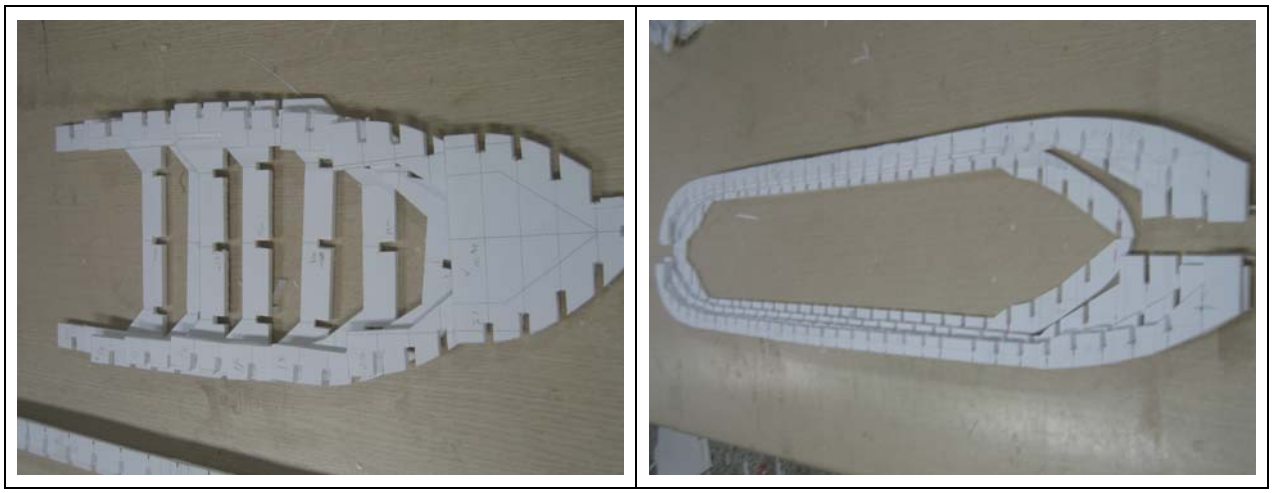


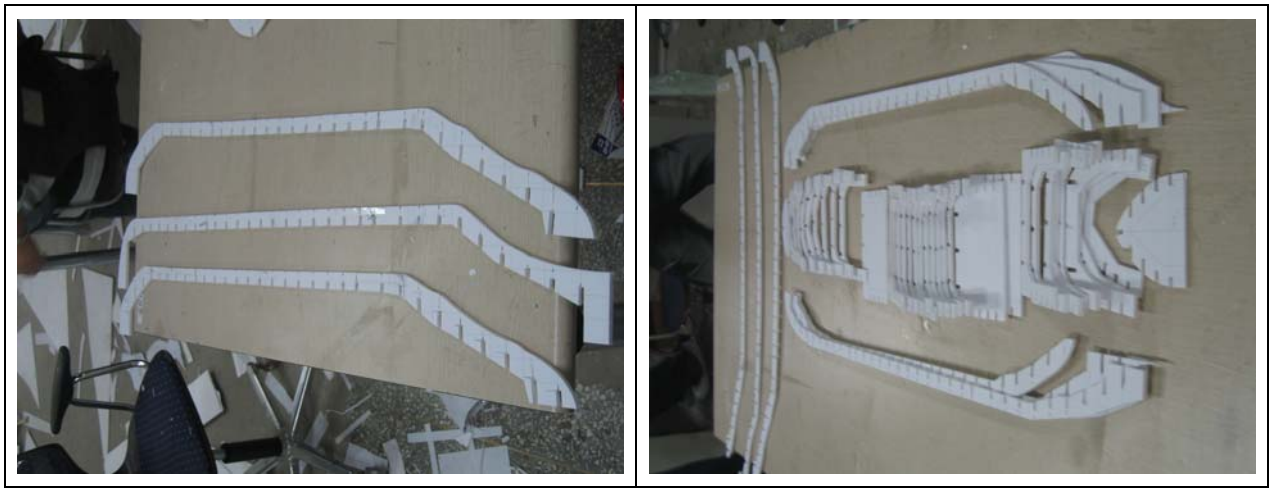
[3] 3.427kg

<그림 28> Nesting

2.2.Cutting

Nesting을 끝낸 후 칼과 자를 이용해 도면에 맞게 부재를 자른다.





<그림 29> Cutting

2.3.부재 조립

Cutting 작업 후 Keel 을 기준으로 모든 부재를 조립한다.

<그림 30> 부재조립

2.4.외판 접합

부재 조립이 끝난 후 1mm 우드락으로 2번 선체 외판을 접합하였다.

<그림 31> 외판접합

2.5.방수처리(PUTTY AND PAINTING)

선체 외판에 Putty를 바르고 사포를 이용해 표면을 매끄럽게 하고 방수처리를 확실히 하기위해 Epoxy를 얇게 골고루 바른 후 Painting 을 2번 하였다.

<그림 32> 방수처리

2.6.추친기 설치

Motor, Propeller, Servomotor 그리고 Battery를 연결하여 설치한다. 설치하는 과정에서 생긴 구멍은 물이 들어올 위험이 있으므로 Glue gun과 Silicone으로 방수 처리를 하였다.

<그림 33> 추진기설치

2.7. Rudder 제작

러더는 선체 제작과 병행하여 설계 및 제작 하였다.

2.7.1 Rudder 개론

타의 목적은 배를 임의의 방향으로 선회 시켜 조종하고 항로의 직진 유지이다. 그리고 Rudder 제작 시 조종성능에 대한 다음 세 가지를 동시에 고려

하여야 한다. 선회반경(Turning ability), 배가 위험을 감지하고 피하기까지 걸리는 시간에 대한 추종성능(course changing ability),그리고 선박이 원하는 방향으로 똑바로 갈수 있는 능력 침로 안정성(course keeping ability)이 있다.

그런데 추종성 및 침로안정성과 선회성은 서로 상반된다.
추종성과 침로안정성을 좋게 하고, 동시에 선회성도 향상시키기 위해서는 타면적을 크게 하여야 한다.
또한 허용 되는 Cb/(L/B)의 값에도 상한이 있으며, 일반으로 Cb/(L/B)<=0.15 정도이나, 타면적을 극단적으로 크게 하여 Cb/(L/B)=0.16정도로 한 배도 있다.

2.7.1.1. Rudder의 종류

1) 균형 타 (balanced rudder) (그림 10.20)

타 기둥(rudder post)이 없고 타면의 일부가 회전축의 앞쪽에 있다.

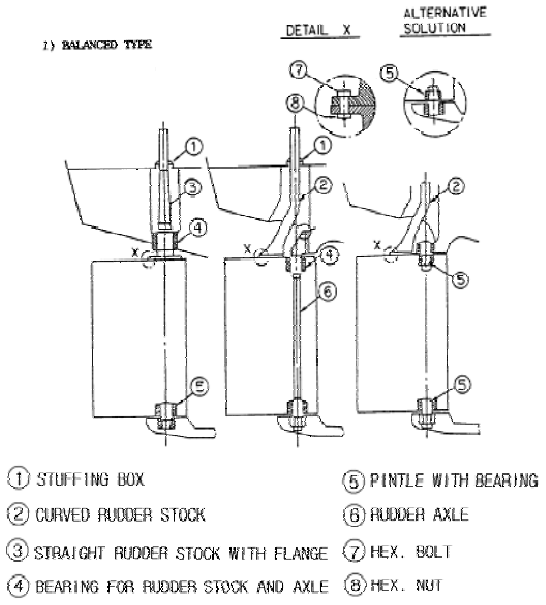
장점

타 압력 중심과 회전중심이 가까워 회전 모멘트가 작아지므로 조타기관의 출력이 작아지며 rudder stock에 가해지는 bending moment도 작아진다.

단점

rudder stock 앞쪽의 타면이 회전할 때 반작용을 일으키므로 선수 유지에

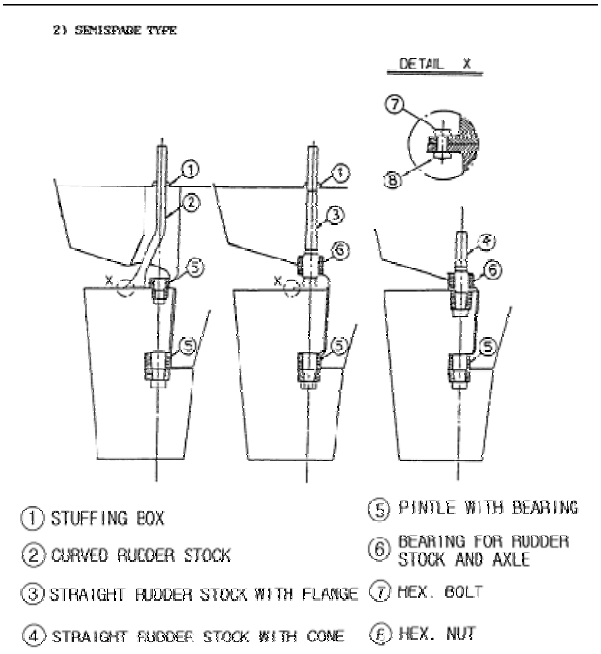
힘들다.



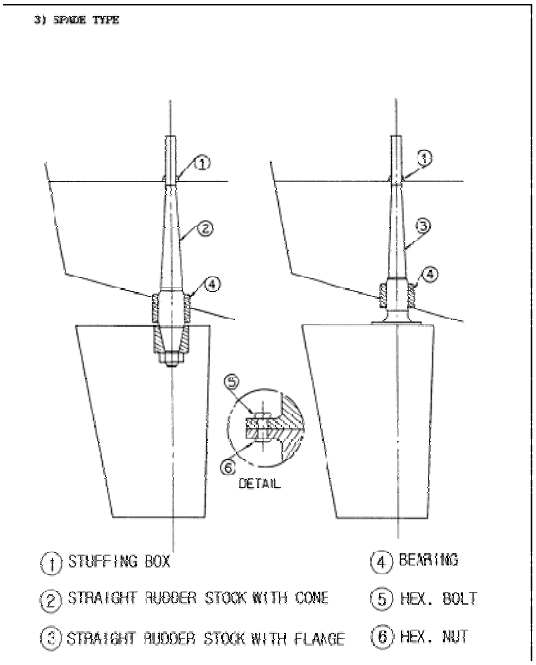
<그림 34> 균형 타(balanced rudder)

2) 반 스페이드 타(semi-spade rudder) (그림 10.21)

rudder horn이 있는 타로서 타 상부는 불균형타로, 하부는 균형타로 되어 있으며 근래에는 많은 배에 채택되고 있다.



<그림 35> 반 스페이드 타(semi-spade rudder)



<그림 36> 스페이드 타(spade rudder)

2.7.1.2. Rudder 면적 결정

1) 실적선의 범위

선박의 길이(L_{BP}), 흘수(d), 선종, 선미 LINES 등을 토대로 실적선, 사양서, 선주 요구 등을 참조하여 AREA를 결정한다.

- rudder area ratio 및 rudder balance ratio는 아래와 같이 결정한다.

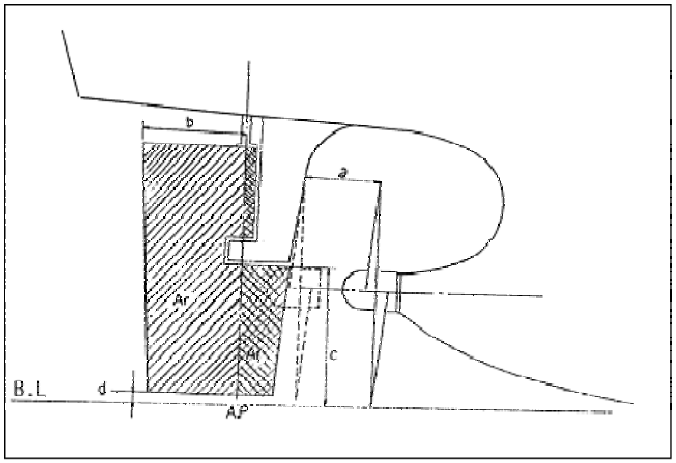
<표 12> rudder area ratio 및 rudder balance ratio

	일반화물선	어 선
RUDDER AREA RATIO ($A_R/(L\times d)$)	1.4 ~ 1.7 %	2.5 ~ 2.7 %
RUDDER BALANCE RATIO (A_f/A_R)	22 ~ 24 %	23 ~ 25 %

여기서, A_R : total rudder area (rudder horn 제외)

A_f : fore rudder area (그림 10.24 참조)

- aspect ratio는 2 정도가 적당하다.



<그림 37> Rudder 배치 그림

2) DnV Rudder 면적 추정식

$$\frac{A_R}{L \cdot d} = 0.01 + 0.5 \cdot \left(\frac{C_b}{L/B} \right)^2$$

여기서, 타의 면적(A_R)은 RUDDER HORN이 있는 경우, HORN을 포함한 면적을 의미한다.

a : 프로펠러와의 최소거리(minimum tip clearance) 이상이어야 한다.

minimum tip clearance 규정

DNV : $a \geq 0.2R(m)$

a : 0.7 R 에서의 간격임

a : 프로펠러 반경

b : 선체 후부 격벽(aft end bhd)을 벗어나지 않도록 한다.

c : 프로펠러 발출(propeller removal)시 간섭되지 않도록 한다.

d : docking시 타가 손상을 입지 않도록 기선(B.L)과 적당한 간격을 유지하여야 하며,

보통 $400mm$ 정도이다.

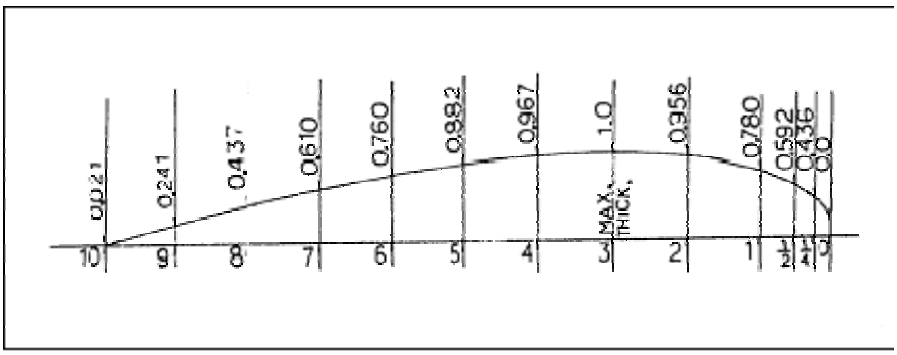
2.7.1.3. 타의 형상(rudder lines)

타의 형상은 선주 요구나 조종시험의 결과가 없을 경우 통상 NACA lines를 사용한다.

선박의 저항면에서는 가능한 얇은 단면(NACA NO. 가 적은 쪽)이 유리하나 강도면에서는 얇은 단면이 불리하므로 타의 강도를 검토하여(pintle 부의 section modulus) 타의 외판이나 casting의 두께가 지나치게 두꺼워지지 않는 범위내에서 단면을 선정하도록 한다.

주로 NACA 0018 ~ NACA 0021 의 범위에서 사용된다.

NACA 단면의 기본 형상은 아래와 같다.(그림 10.25 참조)

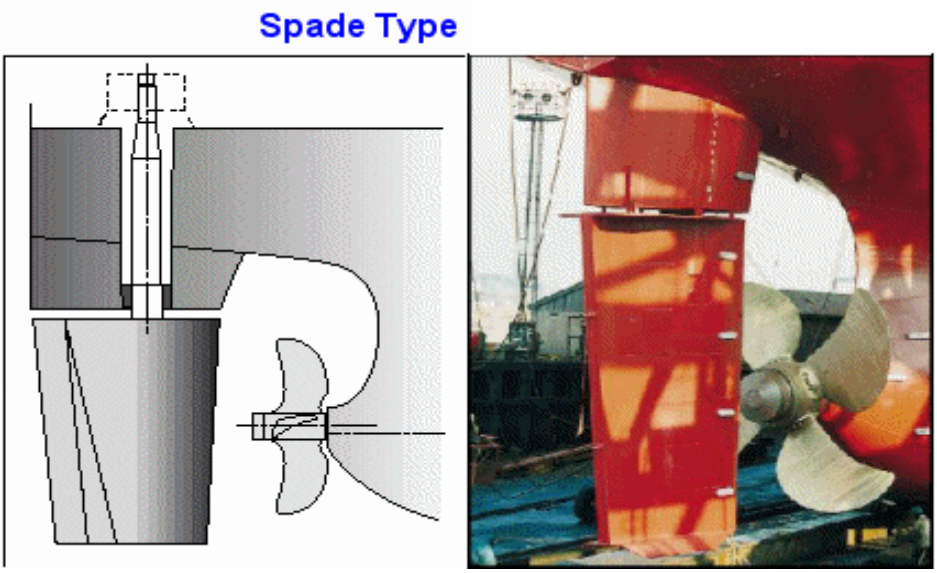


<그림 38> NACA 단면

2.7.2. Rudder 설계

2.7.2.1. Rudder 종류의 결정

우리 조의 러더는 스페이드 타로 선택하여 제작하였다. 그 이유는 Balenced rudder는 rudder stock 앞쪽의 타면이 회전할 때 반작용을 일으키므로 선수 유지에 힘들고 semi-spade rudder는 제작이 용이하지 않아 이번 컨테스트의 주안점이 자동 제어이기 때문에 제작이 용이한 spade rudder를 선택하게 되었다.



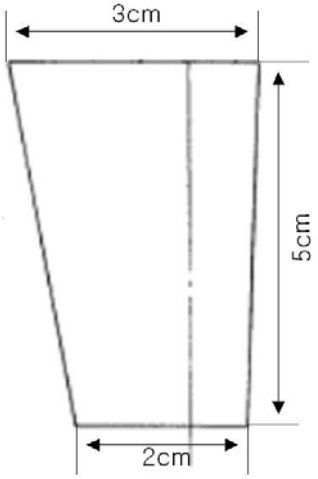
<그림 39> Spade Type Rudder 의 형상 및 사진

2.7.2.2. Rudder 치수의 결정

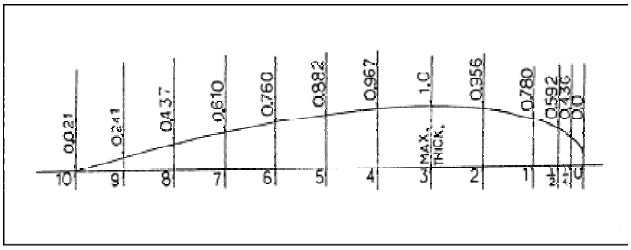
DnV 면적 추정식을 이용한 계산해 본 결과 다음과 같았다.

$$\frac{A_R}{L \cdot d} = 0.01 + 0.5 \cdot \left(\frac{C_b}{L/B} \right)^2 = 0.021777$$

이 수치에서 Rudder의 면적 값을 계산해본 결과 $A_R = 14.48 cm^2$ 임을 알 수 있었다. 따라서 다음과 같은 형상으로 Spade Type Rudder를 제작하기로 하였다.



<그림 40> Rudder 측면도



<그림 41> Rudder 단면

Rudder 단면은 이론에서 살펴보았던 NACA 단면을 출력해 활용하기로 하였다.

2.7.3. Rudder 제작

Rudder가 제작 된 결과

Spade Type으로 제작 된 Rudder의 면적이 $13.75cm^2$ ~ $15cm^2$ 임 을 알 수 있

었고 설계 목표 값인 A_R=14.48㎠ 을 잘 만족함을 알 수 있었다..

3. 제어부 설계

3.1. 하드웨어 설계

3.1.1. 회로이론

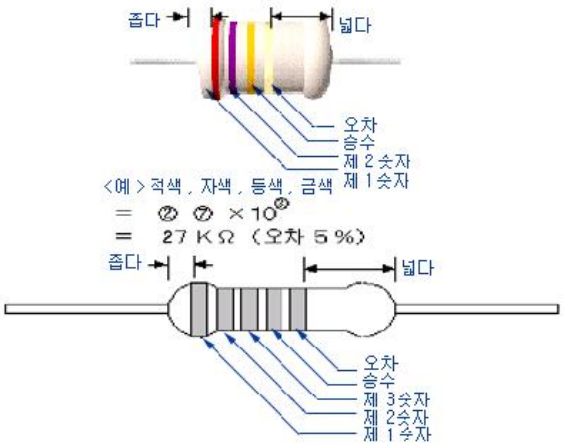
3.1.1.1. 저항기

저항기(Resistance)란 전류의 흐름을 방해하는 기능을 가지고 있습니다. 저항값의 단위는 ohm(Ω:옴)이 사용됩니다. 또한, 1000Ω은 1kΩ(킬로옴), 1000kΩ은 1MΩ(메가옴)이라고 환산하여 표현합니다.

저항기의 종류는 크게 고정저항기와 가변저항기로 나누어지는데 사용하는 재료에 따라 탄소계와 금속계로 분류합니다. 저항기를 사용하는 경우에 중요한 요인은 저항값은 물론이고, 정격전력, 저항값 오차가 있다.정격전력이란 저항체가 견딜 수 있는 소비전력(W:와트)으로, 전력은 전류의 제곱(I^2)×저항(R)으로 구할 수 있습니다. 이러한 저항기가 열을 발생하게 되면 결국 타버리는 경우도 흔히 있으므로 전자회로에서 흔히 사용되는 것으로 1/8W, 1/4W, 1/2W 등이 있습니다. 전자회로의 신호회로(미약전류)에서는 너무 의식할 필요 없이 1/8W로 충분하지만, 전원회로, 발광 다이오드의 전류제어용과 같은 저항기에는 생각보다 큰

전류가 흐르기 때문에 정격전력을 염두에 둘 필요가 있습니다. 정격전력은 예를 들면 12V의 전원전압을 사용하고, 5V에서 동작하는 회로를 동작시키려는 경우를 생각해 보면, 보통은 3단자 레귤레이터 등을 사용하지만, 간단히 저항기만으로 전압을 떨어뜨리려고 한 경우에는 저항기의 저항 값 이외에, 정격전력도 계산해 둘 필요가 있습니다. 이때, 5V에서 동작하는 회로의 소비전류를 모르면 계산할 수 없습니다. 부품의 규격 표에서 조사하거나, 시험 삼아 회로를 만들어 테스터로 측정해 보는 방법 등으로 구하여 보면, 소비전류가 100mA일 때 저항 값은 12V에서 5V로 낮추려 고하면, 저항기에 7V가 걸리므로 7V÷0.1A=70Ω가 됩니다. 이 저항기에서의 소비전력은 70×0.1A×0.1A(또는 7V×0.1A)=0.7W가 됩니다. 계산상으로 구해진 소비전력보다 여유가 있는 정격전력의 저항기를 선택한다. 이 경우 1W가 적당한데, 기본적으로는 소비전력의 2배 정도에 해당하는 정격전력의 저항기를 사용하는 편이 무난합니다.

저항 값의 표시는 숫자로 인쇄하기 위해서는 부품이 작기 때문에 컬러코드(color code)라고 하는 색깔로 표시하고 있는 경우가 많은데, 1/2W 이하의 저항기의 대부분은 컬러코드로 표시하게 때문에 컬러코드를 읽는 법을 꼭 알아 두어야 합니다.



<그림 42> 저항의 예

3.1.1.2. 콘덴서

축전기라고도 한다. 전기용량 C 를 가진 콘덴서에 전압 V 를 가하면, Q =CV의 전하가 축적된다. 또한 이것에 교류전압 v 를 가하면 전하는 충·방전되어, 직류의 경우 전하는 축적되지만 전류가 흐르지 않는다. 이에 대해 교류의 경우는 C dv /dt 인 전류가 흐른다. 교류전압이사인파 교류전압(각주파수 ω)이면, 콘덴서에 흐르는 전류는 그 크기가 1/ωC 의 리액턴스에 흐르는 전류와 같고, 위상이 전압보다 90 ° 빠르다. 2개 이상 콘덴서를 결합할 경우, 병렬결합 때는 합성용량 C는 $C=C_1+C_2$로 되고, C_1, C_2의 어느 쪽보다도 커진다. 직렬결합의 경우에는 합성용량 C는 $1/C=1/C_1+1/C_2$, 즉 $C=C_1C_2/(C_1+C_2)$로 되어 C_1, C_2 어느 쪽보다도 작아진다.

구조는 보통 2장의 서로절연 된 금속 또는 전기 전도도가 큰 도체를 전극으로 하고, 그 사이에 절연체를 넣어 이 사이에서 생긴 정전용량을

이용한다. 이 정전용량 C 의 크기는 전극면적 및 절연된 전극 사이 유전체의 유전율에 비례하고, 전극 사이 거리에 반비례한다. 따라서 실효표면적이 큰 구조가 되게 하거나, 유전율이 큰 유전체를 사용하거나, 또한 절연파괴전압이 큰 유전체를 사용하여 전극간격을 좁게 하면, 용량은 같지만 콘덴서가 소형으로 된다. 콘덴서는 사용전압에 견디는 절연내력을 가지고 있는 것이 필요하며, 이것은 사용 유전체의 절연내력으로 대략 결정되지만, 전극 또는 조립된 구조에 따라 이것보다 저하되는 경우도 있다. 이상적인 콘덴서는 용량 C 만을 가지는 것이지만, 유전체 절연저항이 무한대로 되지 않는 것, 전체 구조상 그 외의 부분에 약간의 누설되는 전류가 있는 것, 유전체에 고주파 전압을 가하면 약간이긴 하지만 그 전력의 일부를 흡수하여 유전 손실을 발생하는 경우도 있는 등의 이유로, 전력의 극히 일부는 콘덴서에서 소비되어, 용량 C에 병렬로 저항 R 를 접속한 것같이 된다. 또한,단자에 인출선 ·전극 등은 고주파수에서는 그 인덕턴스 L 및 저항 r가 영향을 주게 된다. 물론 실용되는 콘덴서는 이러한 영향이 실제로 무시될 수 있는 상태에서 사용되고 있으나, 콘덴서의 성능을 표시함에 있어서는 저항과 리액턴스의 비로 나타낸 $Q = \frac{1}{\frac{1}{\omega \cdot C \cdot R} + r \cdot \omega \cdot C}$ 이라는 양을 사용하며, 이 값이 큰 것일수록 양질의 콘덴서가 된다. 또한 전력손실이 있으면, 흐르는 전류의 위상이 빠른 정도는 90 °보다 δ 정도 작아지고, 이것이 작을수록 양질의 콘덴서이며, 그 좋은 정도를 표시하는데 $\tan\delta \approx \frac{1}{Q}$ 을 사용하는 경우가 많다. 이 밖에 콘덴서로서는 용량, 기타 특성의 안정도, 수명 등이 문제가 된다. 사용하는 유전체에는 많은 종류가 있으며.

이에 따라 분류하면, 공기 콘덴서 ·진공 콘덴서 ·가스입 콘덴서 ·액체 콘덴서 ·운모 콘덴서·종이 콘덴서 ·금속화종이 콘덴서 ·자기 콘덴서 ·유기막 콘덴서 ·전해 콘덴서 등으로 나눈다. 또 정전용량이 변화하지 않는 고정 콘덴서와 변화하는 가변 콘덴서(가변 축전기 또는 베리어블 콘덴서, 간단히 약해서 바리콘이라고도 한다)로도 분류한다. 사용 전압만이 문제가 되는 전압 콘덴서 외에, 사용 전력이 문제가 되는 전력용 콘덴서(kVA 콘덴서)도 있다.

(a) 진공 콘덴서(vacuum condenser) : 유전체를 사용하지 않고 유리와 같은 진공용기 속에서 전극을 대향(對向)시킨 것. 진공 속에서는 글로 방전이 발생하지 않고 또 대기 속의 습도 ·탄산가스 등의 영향이 없으므로 내전압(耐電壓)이 높고 안정하다. 따라서 전극 사이 거리를 좁힐 수 있고 소형이며, 용량이 큰 것을 얻을 수 있다. 또 높은 주파수라도 손실이 적고 안정하므로 송신기와 같은 고주파의 대전력에 적합하여 많이 사용된다. 구조는 고정용량형(동량형)과 가변용량형(가변형)이 있다. 대전력에 사용할 때는 외관쪽을 접지쪽으로 해서 사용하는 편이 좋다.

(b) 공기 콘덴서(air condenser) : 유전체로서 공기 자체를 사용하는 콘덴서. 밀봉하여 건조공기를 충전한 것은 경년변화(經年變化)가 적으므로 표준용으로 사용된다. 또 휴대용 이외의 라디오 수신기의 바리콘(가변콘덴서)에는 가변용량형 공기 콘덴서가 사용된다. 얇은 판으로 된 금속극판이 공기 속의 음향으로 진동하여 용량값이 변동하는 것을 억제하기 때문에, 보통 극판을 상호 결합해서 기계적 강도를 크게 하고 있다.

(c) 금속화종이 콘덴서 : 파라핀 등을 함침(含浸)시킨 얇은 종이의 한쪽 면에 아연 ·알루미늄 등의 금속을 높은 진공 속에서 증발시켜 부착시킨 것(진공증착이라고 한다) 2장을 포개서 감은 콘덴서. MP콘덴서라고도 한다.

이와 같은 종이를 금속화 종이라고 하며, 이 얇은 금속막이 전극으로 사용된다. 따라서 2장을 포갰을 경우, 각 금속막이 대전극(對電極)이 되고, 그 사이에 1장의 종이가 절연물로서 끼워진 콘덴서가 된다. 특징은 전극이 대단히 얇기 때문에 종이의 작은 구멍에서 절연파괴가 일어나도, 그 주위의 금속막이 파괴할 때 전류 때문에 소실되어 절연이 회복된다는 것[自己回復作用], 종이가 1장으로 전극이 얇기 때문에 소형이 된다는 것 등이다.

(d) 전해 콘덴서 : 단순히 전해콘덴서 또는 케미콘(chemical condenser) 이라고도 부릅니다. 이 콘덴서는 유전체로 얇은 산화막을 사용하고, 전극으로는 알루미늄을 사용하고있죠. 유전체를 매우 얇게 할 수 있으므로 콘덴서의 체적에 비해 큰 용량을 얻을 수있습니다. 특징은 극성(플러스 전극과 마이너스 전극이 정해져 있다.)이 있다는 점이며, 일반적으로 콘덴서 자체에서 마이너스 측 리드를 표시하는 마크가 붙어있습니다. 또 가할 수 있는 전압, 용량(전기를 축전 할 수 있는 양)도 표시되어 있습니다. 이 콘덴서는 1μF 부터 수천 μF, 수만 μF 라는 식으로 비교적 큰 용량이 주어지며 주로 전원의 평활회로, 저주파 바이패스 (저주파 성분을 어스등에 패스시켜 회로동작에 악영향을 주지 않는다.)등에 사용됩니다. 단, 코일 성분이 많아 고주파에는 적합하지 않습니다. 즉, 주파수 특성이 나쁘다고 할 수 있죠.

<그림 43> 전해 콘덴서

3.1.1.3. 다이오드

다이오드는 진공관(또는 방전관) 다이오드와 반도체 다이오드가 있으며, 진공관의 경우는 2극관(二極管) 또는 2극진공관이라고도 한다. 셀렌(selenium) 정류기(整流器)와 같은 전자현상을 이용한 이른바 금속정류기 등도 다이오드의 일종이지만, 보통 정류기라고 한다.

3.1.1.4. 트랜지스터

트랜지스터는 기본적으로는 전류를 증폭할 수 있는 부품입니다. 아날로그 회로에서는 매우 많은 종류의 트랜지스터가 사용되지만 디지털 회로에서는 그다지 많은 종류는 사용하지 않습니다. 디지털 회로에서는 ON 아니면 OFF의 2치 신호를 취급하기 때문에 트랜지스터의 증폭 특성에 대한 차이는 별로 문제가 되지 않고요 디지털 회로에서 트랜지스터를 사용하는 경우는 릴레이라고 하는 전자석 스위치를 동작시킬 때(릴레이는 구동전류를 많이 필요로 하기 때문에 IC만으로는 감당하기 어려운 경우가 있음)나,발광

다이오드를 제어하는 경우 등이 있고 트랜지스터는 반도체의 조합에 따라 크게 PNP 타입과NPN 타입이 있습니다. (PNP 타입과 NPN 타입에서는 전류의 방향이 다름.) 마이너스 전압측을 접지로,플러스 전압측을 전원으로 하는 회로의 경우, NPN타입 쪽이 사용하기 쉽습니다.

3.1.1.5. 기판

전자회로를 조립하기 위해서는 부품을 탑재하여 배선할 필요가 있다. 프린트 기판(Printed Circuit Board: PCB)이라고 하는 것이 바로 그것인데, 손수 제작하는 전자회로의 경우는 프린트 기판을 그 때마다 만드는 것은 매우 번거로운 일이다. 그래서, 아래의 사진과 같은 유니버설 기판을 사용하는 방법이 있다. 유니버설 기판은 절연판(유리 에폭시, 종이 에폭시, 베이클라이트 등)에 0.1인치(2.54mm) 간격으로 약 1mm 정도의 구멍이 사방에 뚫려 있다. 기판의 뒷편에는 구멍을 중심으로 2mm 정도의 동박(랜드)이 프린트되어 있다. 사용법은 표면측에 부품을 탑재하고, 부품의 리드를 구멍을 통해 이면측(납땜하는 쪽)으로 내어, 이면측에서 배선을 하는 것이다.

구멍의 간격이 0.1인치이므로, DIP 타입의 IC를 그대로 실을 수 있다(IC를 탑재하기 위해 0.1인치로 하고 있는 것 같다).사진에 보인 것은 유리 에폭시를 사용한 것으로, 녹색을 띠고 있다. 종이 에폭시는 베이지색, 베이클라이트는 묽은 갈색을 띠고 있다. 구멍의 수에 따라 몇 가지 크기가 있으며, 사진의 좌측부터 차례로 55×40홀(기판치수:160×115mm), 30×25홀(기판치수:95×72mm), 25×15홀(기판치수:72×47mm)의 것이다.

3.1.1.6. 인덕터

저항·콘덴서·전자관·트랜지스터·전원(電源) 등과 함께 전기회로를 이루는 가장 중요한 부품 가운데 하나이다. 구리나 알루미늄 등을 절연성 재료로 싸서 나사 모양으로 여러 번 감은 권선(捲線:코일) 가운데 전류의 변화량에 비례해 나타나는 권선이 인덕터이다. 즉, 도선을 감은 코일로 가장 기본적인 회로 부품이자 회로 소자이다.

3.1.1.7 마이크로콘트롤러

CPU는 레지스터, ALU, 멀티플렉서, 디코더 등 여러 가지 디지털 기능을 수행하는 소자들을 조합해서 설계할 수가 있다. 이러한 설계 방식은 어떤 특수한 용도에 맞게 설계할 수 있는 장점이 있다. 그렇지만 CPU는 컴퓨터의 부품 중에서 광범위하게 사용되는 소자이므로 표준화된 하나의 집적회로로 설계하는 것이 유용할 수가 있다. 마이크로프로세서(MicroProcessor)는 한 개의 IC칩으로 된 CPU를 가리킨다. 여러 IC칩으로 설계한 CPU와는 다르게, 마이크로프로세서에서는 직접적으로 레지스터나 ALU 등과 외부 부품을 연결할 수가 없으며, 다만 이 칩의 단자를 통해서만 정보전달이 가능하다. 또 마이크로프로세서는 CPU의 모든 내용이 하나의 조그만 칩 속에 내장됨으로 해서, 가격이 훨씬 싸지며 부피가 줄어든다는 중요한 장점이 있다.

저렴한 가격과 작은 크기로 인해 마이크로프로세서가 이전에는 경제성이 없던 구조를 설계가 가능하게 해줌으로써 컴퓨터 설계기법에 있어 변혁을 가져오게 했다. 마이크로프로세서는 다양한 방면에 응용되고 있다. 범용 컴퓨터의 CPU로나 특수용 컴퓨터의 프로세서로 뿐만 아니라, 교통신호 제어,

개인 가정용 컴퓨터, 계측 제어기기, 사업용 업무처리 등 다양하게 사용되고 있다

3.1.1.8. 마이크로콘트롤러의 종류

♦ AVR 마이크로 프로세서

①AVR 마이크로 프로세서란?

미국 아트멜(Atmel) 사의 AVR 마이크로 컨트롤러는 Single 싸이클 명령 실행 구조를 갖는 RISC이다. 또한, 효율적인 I/O포트 구조와 각 모델에 따라서 내부 발진회로, 타이머, 시리얼 통신(UART), SPI, AD변환기, 풀업저항(Pull-UP 저항), 펄스폭변조(PWM)제어, 아날로그 비교기 그리고, 와치독(Watch dog) 타이머등이 내장 되어 있다. AVR마이크로 프로세서의 명령어들은 C언어나 기계어(어셈블리)로 개발할 때 프로그램의 크기를 최적화하도록 되어 있다. 또한, AVR 시리즈에는 플래쉬 메모리(Flash Memory)가 내장되어 있어 단 시간 내에 최소의 비용으로 개발하려는 분들에게 매우 적합한 마이크로 컨트롤러이다[4].

②AVR시리즈의 특징

○ 8-비트 RISC(Reduced Instruction Set Computer) 구조로 명령어가 간단하며 동작 속도가 빠르다

○ 1MHz당 약 1MIPS(Million Instruction Per Second)의 성능을 보인다.

---

[4] 네이버 블로그. 2006-09-17. http://blog.naver.com/love4youkr?Redirect=Log&logNo=70004087388/.

○ 소비 전력이 적다.

○ 10 비트 ADC 내장하고 있다.(일부 패밀리)

○ 다른 마이크로 콘틀롤러에 비해 큰 SRAM을 가지고 있다. 예) AT90S8515(512B), Mega103(4KB)

○ Flash memory의 내장으로 프로그래밍이 용이하다.

○ EEPROM을 내장하고 있어서 데이터 백업이 가능하다(EEPROM의 크기는 SRAM과 동일)

○ C언어에 최적화된 설계(실제로 칩 설계와 동시에 C 컴파일러 설계하였다)

○ UART, SPI(Serial Pheriperal Interface), PWM(Pulse Width Modulation) 등을 내장하고 있다.

○ 8비트 및 16비트 타이머를 내장하고 있다.

AVR은 Atmel사의 고밀도 비휘발성 메모리 기술을 기반으로 제조된다. 칩 내에 다운로딩 가능한 플래시 메모리는 비휘발성 메모리로서 SPI 직렬 인터페이스 방식이나 기존의 프로그래밍 장치를 이용해 반복 프로그램 할 수 있는 프로그램 메모리를 RISC 8비트 MCU내부에 구현함으로써 다양한 응용분야에 적용할 수 있는 매우 강력한 마이크로 콘트롤러이다. AVR은 어셈플러 및 C언어를 이용하여 프로그래밍 할 수 있으며, 프로그램 개발에 필요한 툴 등이 통합환경으로 제공되고 있다.

③AVR시리즈의 개발환경

AVR은 내부의 플래쉬 메모리와 EEPROM에 데이터를 써넣을 수 있도록 SPI 인터페이스를 제공한다. SPI는 오직 3라인을 이용한 통신 방법으로 MOSI(Master Out Slave In), MISO(Master In Slave Out), SCLK(SPI CLOCK) 시그널을 이용한다. Motolora에서 개발되었으며 Master와 Slave가

SCLK에 동기하여 데이터를 교환하는 방식이다. 플래쉬 메모리를 억세스하기 위해서는 AVR의 Reset 핀을 low로 한 상태에서 앞의 세 시그널을 이용하여 데이터를 읽고 쓰기가 가능하다. 즉, AVR과의 SPI 인터페이스를 맞추어주기만 하면 내부 플래쉬 메모리의 액세스가 가능하다는 것이다. 이는 AVR의 프로그래머를 싼 값으로 구현하게 하는 동기가 된다[5].

3.1.1.9. A/D 컨버터

·● AD 컨버터란

자연계의 여러 가지 신호는 대부분 신간에 따라 아날로그적(연속적)으로 변화한다. 이때 아날로그라는 것은 연속적으로 변하는 물리적인 변화를 수치적으로 표현하는 방법으로 전압의 변동이나 시간의 변화와 같은 것은 값이 단절되지 않고 연속되기 때문에 이러한 값을 표현하는 방법으로 이러한 방법이 사용된다.

우리는 이 신호들을 시간에 따라 그 크기를 알아내야 할 필요가 있다. 또 그 값들을 컴퓨터를 통해 처리하고 분석하기도 한다. 이 아날로그 신호를 컴퓨터에서 처리할 수 있는 디지털 값으로 변환해 주는 장치가 A/D 컨버터 이다.

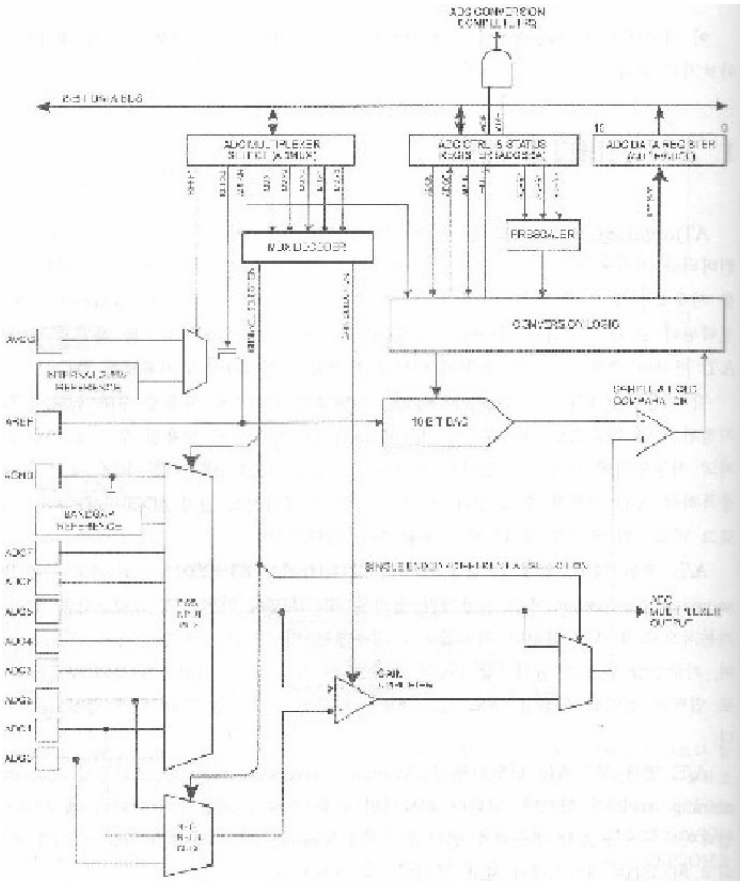
예를 들면 음악 데이터를 디지털 기록 미디어인 CD에 기록할 때에 아날로그 신호인 음성을 디지털 신호로 변환해야 하는 데 이 처리를 하는

---

[5] 로보블럭몰. 2006-09-18. http://www.roboblock.co.kr/info/info2-12.htm/.

것이 바로 A/D 컨버터이다. 디지털로 기록된 음악 CD를 재생할 때에는 A/D 컨버터와는 상반되는 처리를 하는 D/A 컨버터를 이용해 디지털 신호를 아날로그 신호로 바꾸게 된다.

<그림 44> A/D 컨버터



<그림 45> A/D 컨버터의 블록 구성도

•Atmega8535의 A/D 컨버터

ATmega8535에는 총 4개의 8비트 입출력 단자가 있다. 각각은 포트 A, 포트 B, 포트 C, 포트 D이고, 각 포트는 8비트이므로 각각 8개씩 총 32개의 입출력을 지원한다. 입출력은 0또는 1로, 0일때 0v전압을 출력하거나 입력 받고, 1일때 5v전압을 출력하거나 입력 받는다.

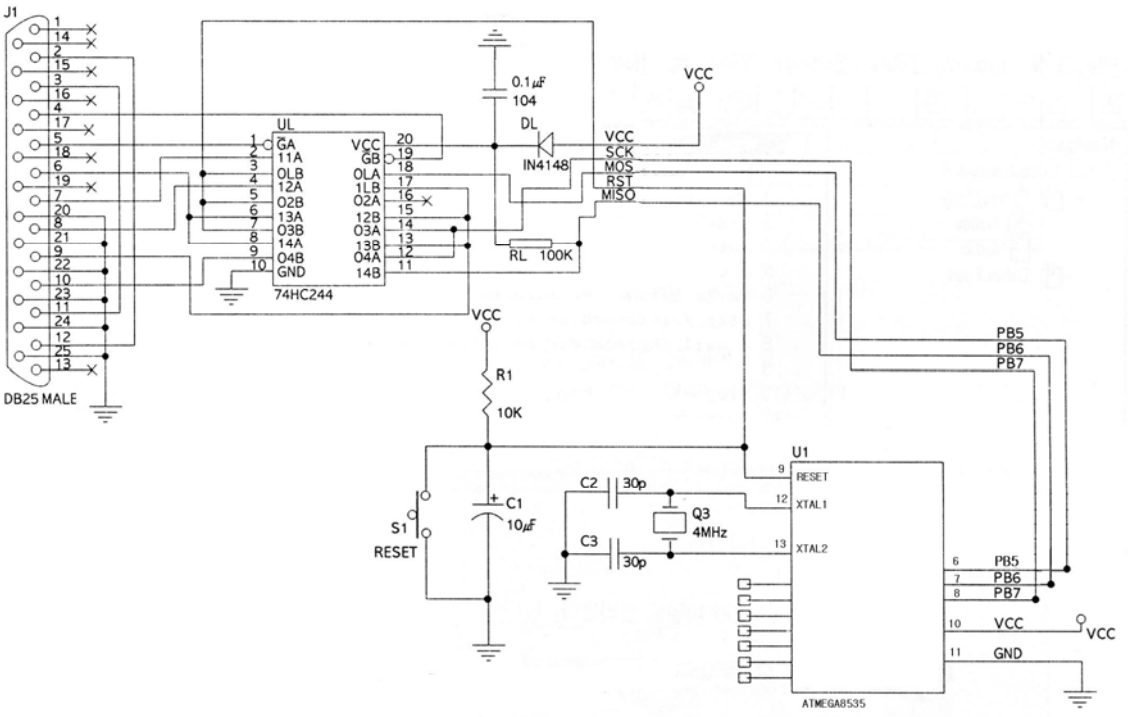
다양한 전압을 아날로그로 출력하기 때문에 avr의 AD컨버터를 통해서 아날로그 형태의 정보를 디지털로 입력 받아야 한다. avr의 ad컨버터는 0~5V까지의 전압을 총 10비트로 받을 수 있다. 즉 2^{10} 총 1024단계로 입력 받을 수 있다.. AD컨버팅이 가능한 핀은 Port A의 0~7번 핀이다. 만약

신호가 작아서 전압이 낮다면 opamp를 통해서 전압을 증폭시켜야 한다. 증폭된 전압을 포트A의 AD 컨버터로 입력 받고 입력된 전압에 따라서 모터를 작동시키기 위해 포트에서 출력을 시키면 5v전압이 나온다.

3.1.2. ROM WRITER 설계

Rom Writer Cable 란 PC에서 작성한 소프트웨어를 MCU로 기억하게 하는 장치이다.

먼저 아래의 그림은 수업시간에 주어진 회로도이다.



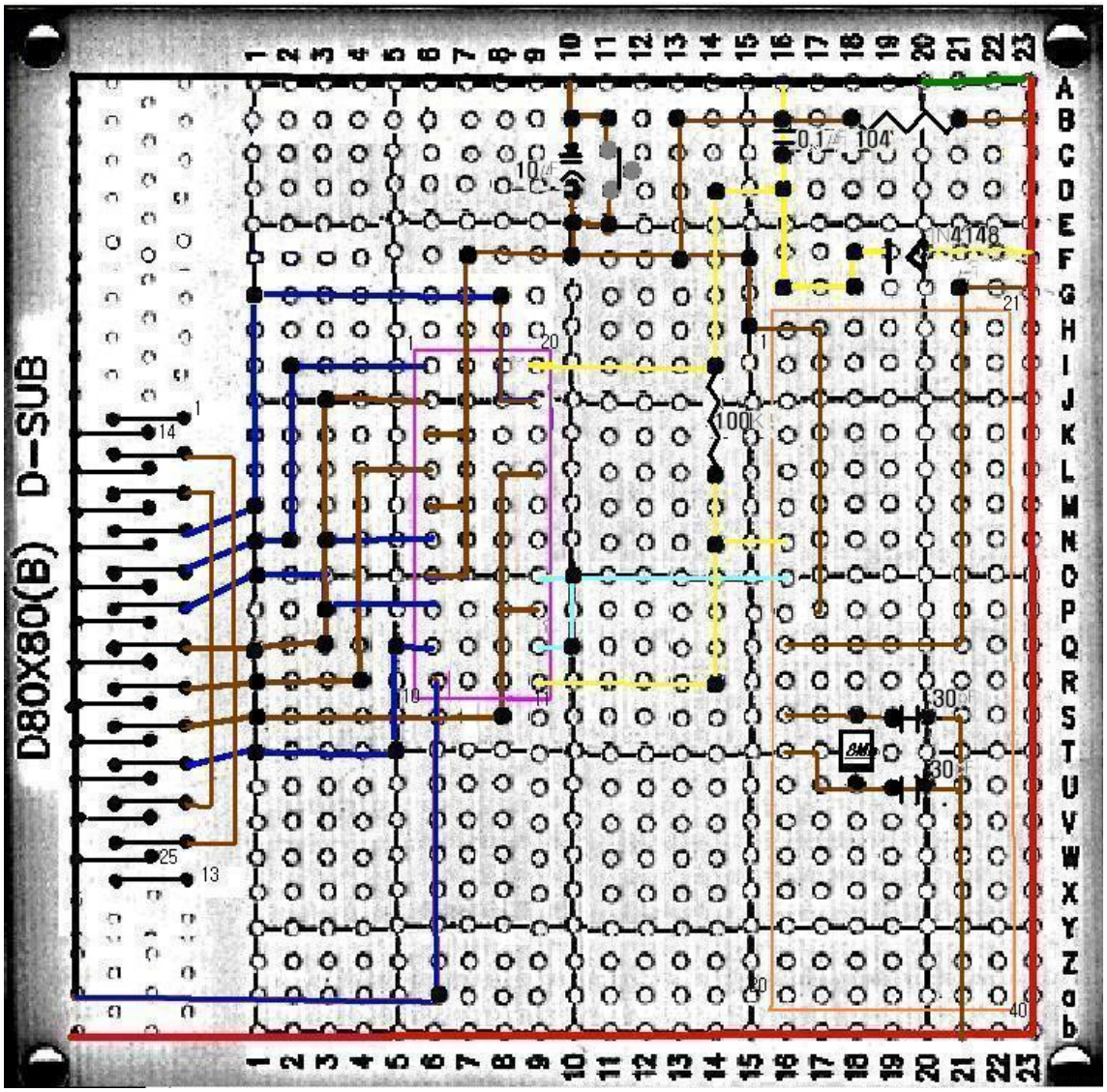
<그림 46> Rom Wrighter Cable

회로도에서 DB25 MALE 은 케이블 꽂는 단자이고, 74HC244는 장치들 사이의 처리속도 차이를 줄여주는 버퍼이다.

PC에서 데이터를 25핀 프린터 케이블을 통해 받아 B번 포트의 5번, 6번, 7번 포트를 이용해서 입력 받는다.

스위치는 보드 전체의 RESET을 위한 것이고 CRYSTAL과 CAPICITOR가 회로의 기본적인 구동을 위해 쓰였다.

다음은 이 회로도를 기반으로 실제로 기판을 제작할 설계도이다. 이 설계도는 주어진 회로도를 바탕으로 더욱 편리하게 제작하기위해 직접 설계한 설계도이다.



<그림 47> Rom Writer Cable 설계도

왼쪽의 작은 분홍색의 직사각형은 버퍼이고, 왼쪽의 주황색 직사각형이 MCU이다. 검은색선이 접지를 위한 선이고, 붉은색 선이 전원공급을 위한 선이다.

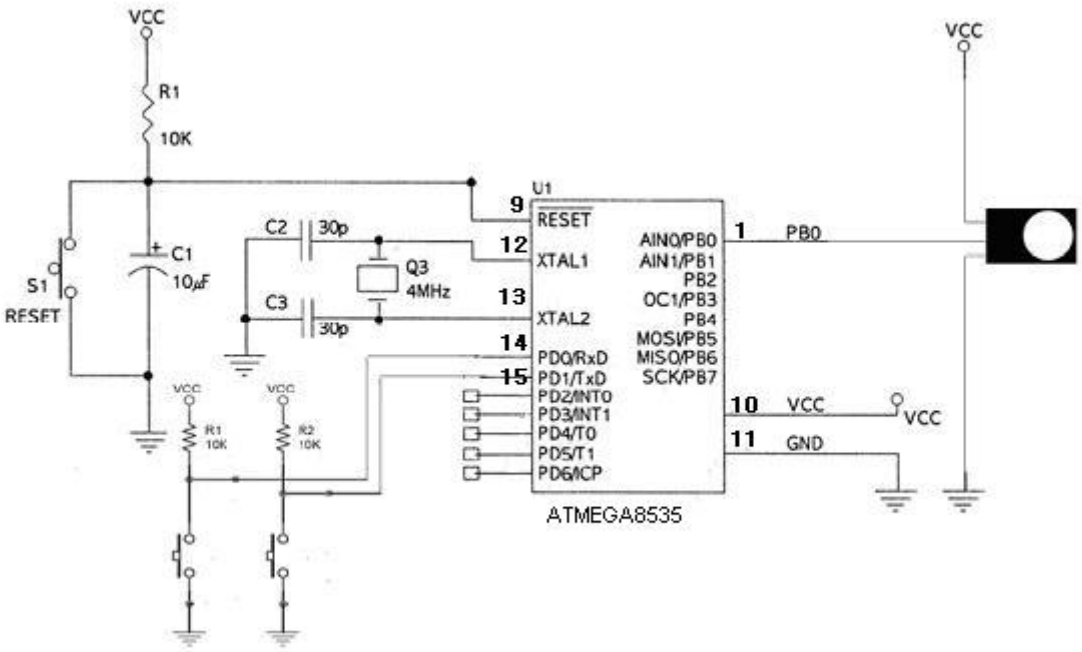
이 Rom Writer Cable을 만들기 위해서는 다음과 같은 재료들이 필요하다.

25Pin D-Sub 커넥터 수놈, 25Pin D-Sub 커넥터용 케이스,74HC244,세라믹 콘덴서 0,1 μF(104),

다이오드 IN4148,저항 100K,프린터 케이블, 74HC244 용 20pin 소켓

3.1.3. SERVO MOTOR 설계

SERVO MOTOR



<그림 48> SERVO MOTOR 회로도

회로를 보면 회로의 RESET과 기본 구동을 위한 연결을 AVR의9,12,13번에 연결어 있다. SERVO MOTOR의 출력신호를 포트B의 0번에 연결되어있다. SERVO MOTOR와 기판은 CONNETER를 이용해 연결한다,

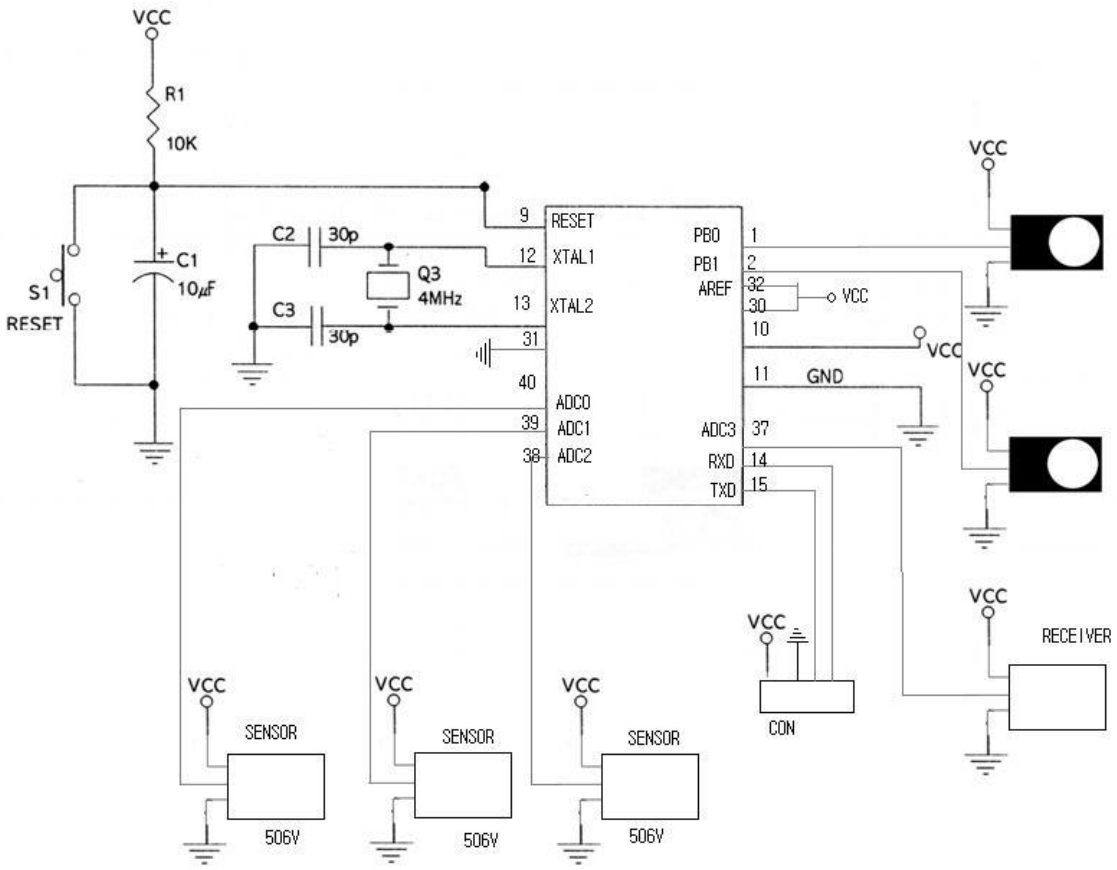
SERVO MOTOR는 빠른 응답과 넓은 속도제어의 범위를 가진 제어용 전동기로, 그 전원에 따라 DC SERVO MOTOR와 AC SERVO MOTOR로 분류된다. AC SERVO MOTOR의 대부분은 3상SERVO MOTOR이다. 이것들은 정지·시동·역전 등의 동작을 반복하므로, 방열효과를 좋게 하거나, 동작의 변화가 빨라지도록 설계상 고려되어 있다. SERVO MOTOR는 제어신호에 따라 운전되며, 그러기 위해서는 제어신호를 받아 이것을 증폭하여 SERVO MOTOR를 구동하는 장치가 필요하며, 이것을 서보 증폭기라고 한다.

선체 제작에서는 방향을 조절하기 위해 RUDDER를 좌우로 움직이는 역할을 SERVO MOTOR에서 맡게 된다.

3.1.4. CONTROLLER 설계(1차)

실제로 선박의 자동제어를 하기위해 필요한 CONTROLLER의 설계이다. 자동제어를 위해서는 먼저 초음파센서를 이용해서 장애물과 선박과의 거리를 측정해야 한다. 측정된 값을 MCU에서 받아들여서 센서에서 아날로그자료를 디지털 자료로 변환하여 적절한 수치를 계산한 뒤 그 수치에 따라 SERVO MOTOR와 변속기에 값을 출력해 주어 선박이 방향을 선회하고 속도를 제어하게 된다.

이러한 CONTROLLER를 만들기 위해 적절한 회로도를 구상하였다.



<그림 49> controller 회로도

회로도를 보면 먼저 3개의 센서를 ADC0, ADC1,AADC2에 연결하여 입력을 받는다. 이 자료를 MCU에서 계산하여 PB0, PB1에 연결된 SERVO MOTOR와 변속기에 출력 값을 주어 제어한다.

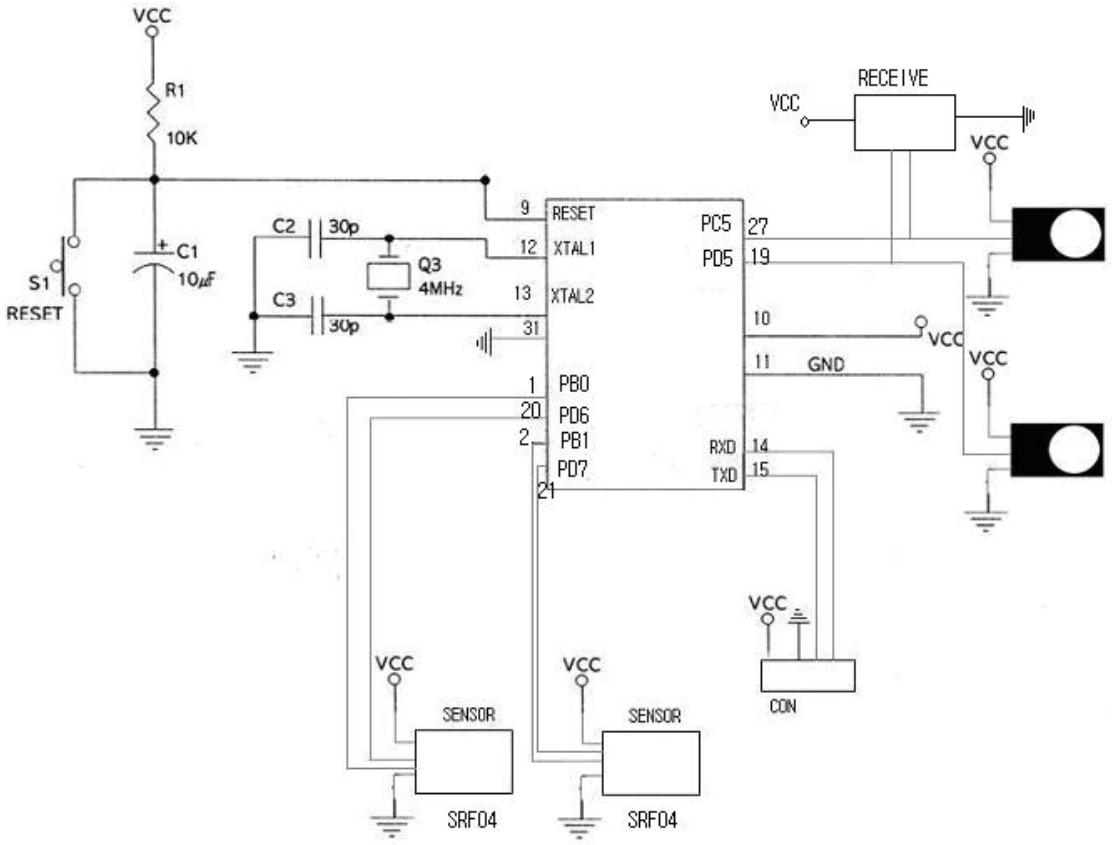
RS232케이블을 통해 MCU에 입력되고 있는 값을 PC를 이용해 확인해보기 위해 RXD와 TXD를 커넥터에 연결한다.

마지막으로 수신기에서 자료를 받고 있는지 여부를 측정하기 위해 입력 값

이 ADC3으로 들어가도록 연결한다. 수동제어를 위한 리모컨이 켜질 때 수신기의 값이 변하는 것을 찾아내어서 수동제어 리모컨이 켜지면 MCU의 PB0와 PB1에서 더 이상 출력 값을 주지않게 프로그래밍 한다. 이러한 방법으로 수동 리모컨을 켜는 것 만으로도 자동/수동 제어를 손쉽게 전환할 수 있다.

3.1.5. CONTROLLER 설계(2차)

1차 컨트롤러 설계대로 제작을 하다가 문제가 생겨 컨트롤러의 설계를 다시 하게 되었다. 먼저 소프트웨어를 통한 자동/수동 제어에 실패하여 스위치를 사용해 자동/수동 제어를 하게 되어서 기판이 바뀔 수 밖에 없었고, 여러 번의 시행착오 끝에 같은 신호를 받아서 작동되는 전기부품 일절은 모두 같은 전원과 그라운드에 연결되어 있어야 한다는 것을 경험적으로 깨달았기 때문이다. 또 센서에서도 기존의 센서의 성능이 좋지 않아서 같은 초음파 센서이지만 사용법이 다른 SRF04라는 모델의 센서를 설치하여 다시 설계 하였다.



<그림 50> 2차 controller 회로도

1차 회로와 비교를 해보면 먼저 센서가 변했다. 1차 설계 때 사용한 센서는 센서에 전압만 공급해주면 다른 작업 없이도 바로 아날로그 신호 값이 나오지만 이 값을 ADC로 받아들여서 변환해주는 작업이 필요하다. 그러나 새로운 센서는 굳이 PORT A에 연결할 필요 없이 아무 포트나 두개의 포트를 할당하여 하나는 센서에 일정한 주기의 펄스파를 주고 하나의 포트에서는 그 값을 읽어 들이는 것이다. 또 서보모터와 변속기 수신기를 연결하는 선이 크게 바뀌었다. 앞에서도 언급했듯이 하나의 전원이 모든 전기부품에 같이 연결되어야 하기 때문에 종전에 서보모터와 변속기에서 신호선만을 기

판에 연결해준 것과 달리 전원선, 그라운드선, 신호선 모두를 기판에 같이 연결해주어 공통된 전원과 그라운드를 갖게 하였다.

3.2. 소프트웨어

3.2.1. C언어

C언어는 1972년 미국 벨 연구소의 데니스 리치가 설계 ,DeC사의 소형 컴퓨터인 PDP-11상에 최초로 탑재 되었다. 운영체제 시스템으로 유명한 유닉스도 90% 이상이 C언어로 제작 되었다.

C는 원래 OS등의 시스템 기술 언어로서의 성격을 갖고 있었으나 그 우수한 기능이 알려지게 되면서 현재는 프로그램 언어로서 널리 쓰여지고 있다. 특히 마이크로 컴퓨터의 보급에 따라 많은 기종에 C가 이식되어 지금은 전문가뿐 아니라 일반 사용자에게도 널리 보급되어 있다.

이전에는 빠른 실행 속도를 요구하는 프로그램은 어셈블리어로 개발되어 왔으나 현재는 그 대부분이 C로 대치 되었다. 어셈블리어는 전체 프로그램 중에서 고속의 실행 속도가 필요한 모듈에만 부분적으로 이용되고 있다. 물론 어셈블리어는 앞으로도 계속 사용되겠지만 생산성 면에서 C의 우위는 확고부동하다.

또한 지금은 C언어의 기능을 강화한 C++가 더 많이 쓰이고 있다. 윈도우용 응용프로그램을 개발하기 위한 Visual C++ 등의 툴도 있는데 이러한 C++를 사용할 때도 C에 대한 지식이 필요하다.C++가 C에 대한 지식을 전제로 한 언어이기 때문이다. 또한 C는 정보 처리 기사 시험의 대상 언어로도 지정되면서 그 중요성은 차츰 커지고 있다.

C언어는 Algol언어를 기원으로 하고 있다. Pascal도 Algol에서 발전한 것이기 때문에 C와 Pascal은 비슷한 구조로 되어 있다. 그러나 Pascal은 교육용 언어로서 언어의 완전성을 추구하는 데 반해 C는 실무용 언어로서 실무 개발자가 이용하기 쉽도록 언어의 기능성을 중요시 한다. 그 결과 C는 강력하기는 하지만 작은 실수를 용납하지 않는 언어가 되었다.

그러나 근대적인 구조화 프로그램의 개념을 구현하고 어셈블리어에 가까운 상세한 기술을 가능하게 하는 C언어는 현재 컴퓨터 업계의 주요 언어로서의 지위를 차지하고 있다.

3.2.2. Code Vision

CodeVisionAVR C 컴파일러는 가격에 비해 성능이 우수한 컴파일러로 다음과 같은 특징이 있다

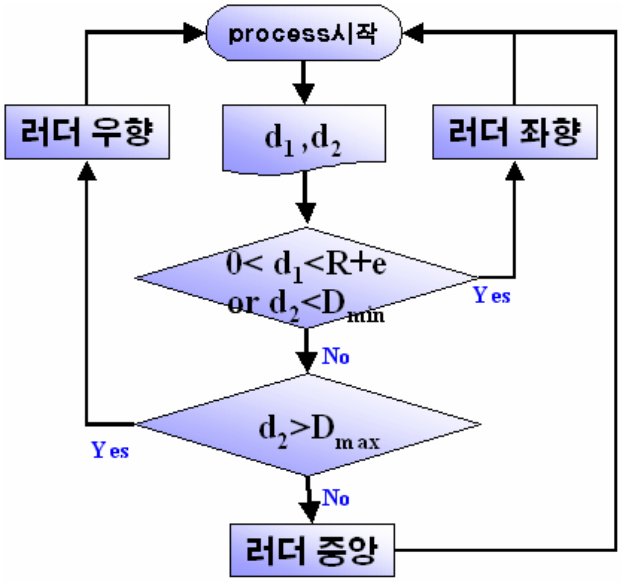
- 편리한 통합 환경 : 소스 에디터, 컴파일/링크, Flash-ROM/EEPROM 다운로드, 통신 프 로그램 등이 함께 제공 됨.
- 자동 소스 코드 생성 기능(CodeWizardAVR) *1
- 풍부한 라이브러리 함수 제공 : 통신(UART), LCD 표시, 온도, 시계, A/D 소자함수 제공.
- 편리한 FLASH, EEPROM, SRAM 변수 관리
- 포트 및 변수의 비트 제어 기능 *1
- 인 라인 어셈 기능 제공 *1
- 저 가격. (20 만원 대)
- 무료 평가 버전 사용 가능(용량 제한 및 일부 함수 사용 제한)

• IAR C 컴파일러에는 없음(또는 불편)

3.2.3. 알고리즘

2.2.3.1. 주어진 경로로 주행

주어진 경로로 주행 시 벽과 간격의 멀어지고 가까워짐에 대한 러더 움직임을 알고리즘으로 표현

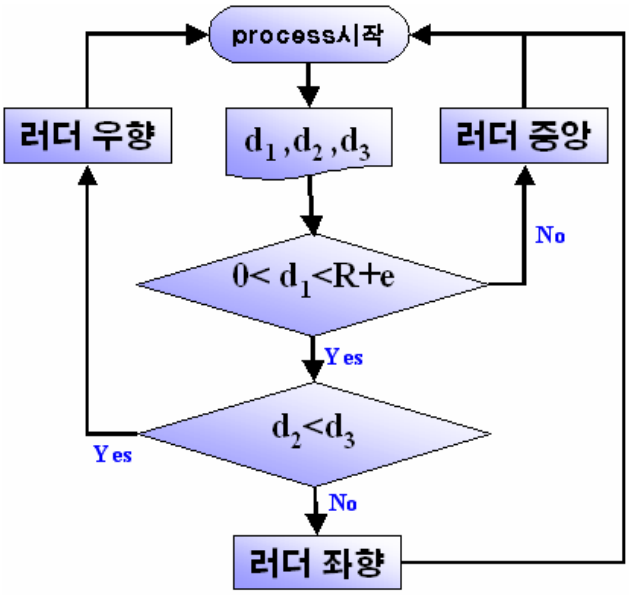


d_1	전방센서입력값
d_2	우측센서입력값
D_{min}	측벽거리하한
D_{max}	측벽거리상한
R	선회반경
e	margin

<그림 51> 주어진 경로를 주행하는 알고리즘

2.2.3.2 장애물 회피

장애물 감지 후 회피움직임에 대한 러더 움직임을 알고리즘으로 표현



d_1	전방센서입력값
d_2	우측센서입력값
d_3	좌측센서입력값
R	선회반경
e	margin

<그림 52> 장애물 회피 알고리즘

4. 제어부 제작

4.1. 하드웨어

본격적으로 선박을 제어 할 CONTROLLER를 만들기 이전에 전기소자와 회로에 대한 이해를 돕고 CONTROLLER 제작을 위하여 Rom Writer, led & 가변저항, Servo Moter, RS232 통신케이블의 제작을 수행하였다.

4.1.1. 롬라이터 제작

롬라이터 제작에 필요한 준비물은 다음과 같다.

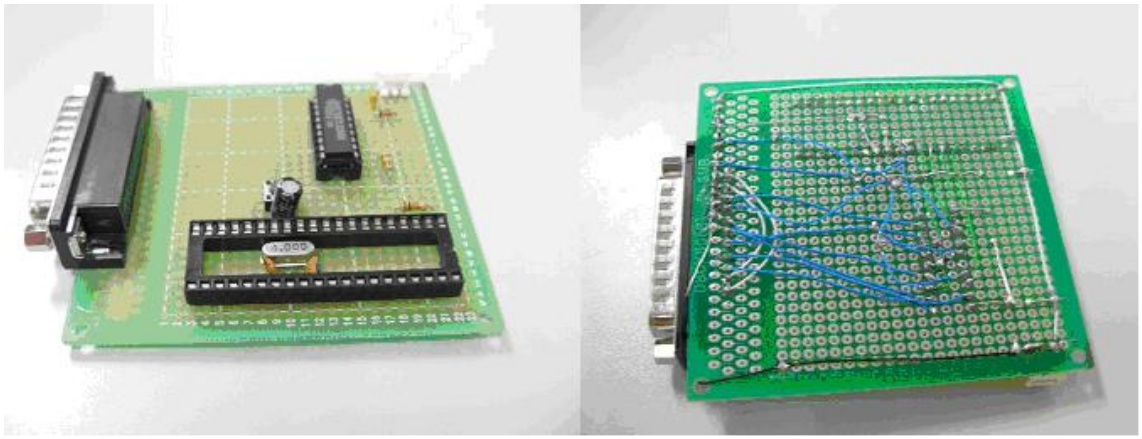
* 준비물 : 74HC244, DB25 MALE, Capacitor 0.1㎌,

Diode IN4148, Resistance 100K, D-SUB기판, 40pin 소켓,

4Mhz 크리스탈, 스위치

<그림 53> Rom Writer 준비물

<그림 54> Rom Writer 제작사진

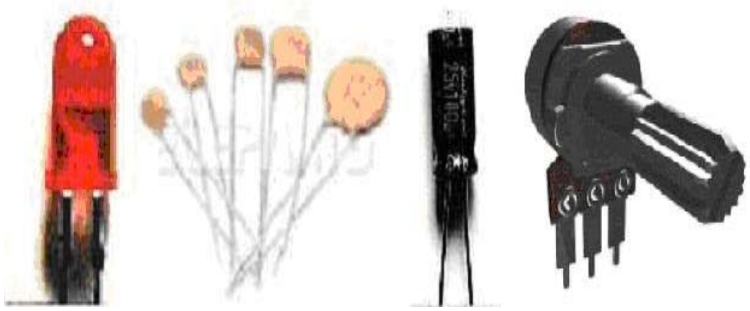


<그림 55> 제작된 Rom Writer의 전면 & 후면

4.1.2. led & 가변저항 제작

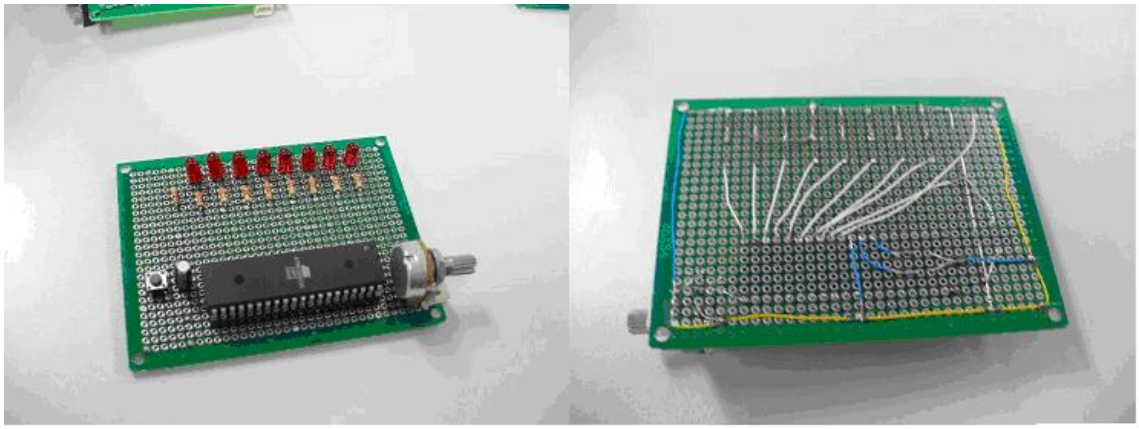
LED와 가변저항을 결합한 회로의 준비물은 다음과 같다.

- 준비물 : LED 8개, 330Ω 저항 8개, ATMEGA8535, 30pF 캐퍼시터 2개, 4Mhz 크리스탈, 10μF전해콘덴서, 스위치, 10KΩ 저항, 가변저항



<그림 56> LED & 가변저항 준비물

<그림 57> LED & 가변저항 제작과정

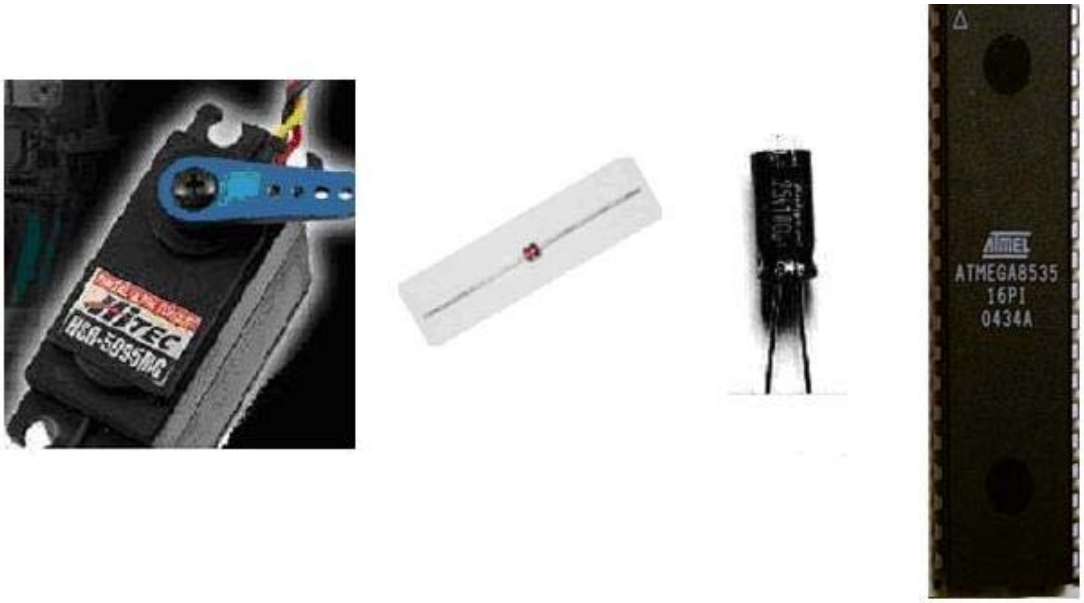


<그림 58> 제작된 LED & 가변저항의 전면 & 후면

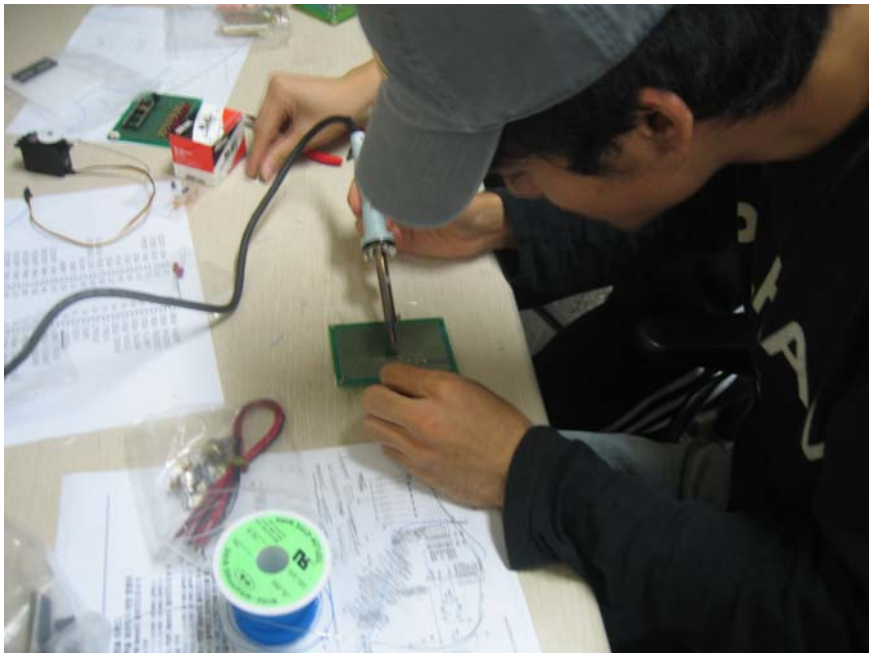
4.1.3. Servo Moter 제어기 제작

Servo Moter는 Rudder를 돌리는 부품으로 이번 과제 중에서도 이번 프로젝트와 가장 연관이 깊은 부분이었다. 따라서 Servo의 작동여부는 이번 프로젝트의 성사를 결정짓는 중요한 부분이었다.

* 준비물 : ATMEGA8535, 30pF 캐퍼시터 2개, 4Mhz 크리스탈,
10μF전해콘덴서, 스위치 3개, 10KΩ, 저항 3개, 서보모터



<그림 59> Servo Moter 제어기 준비물

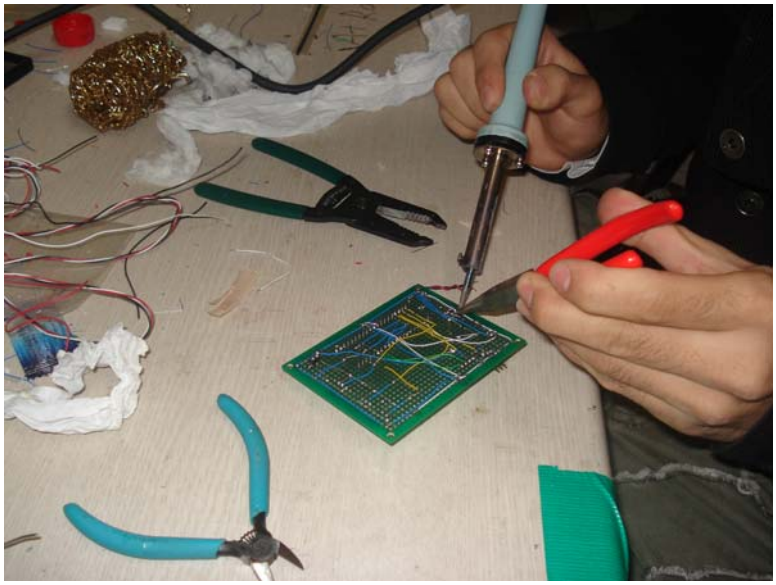


<그림 60> Servo Moter 제작 과정

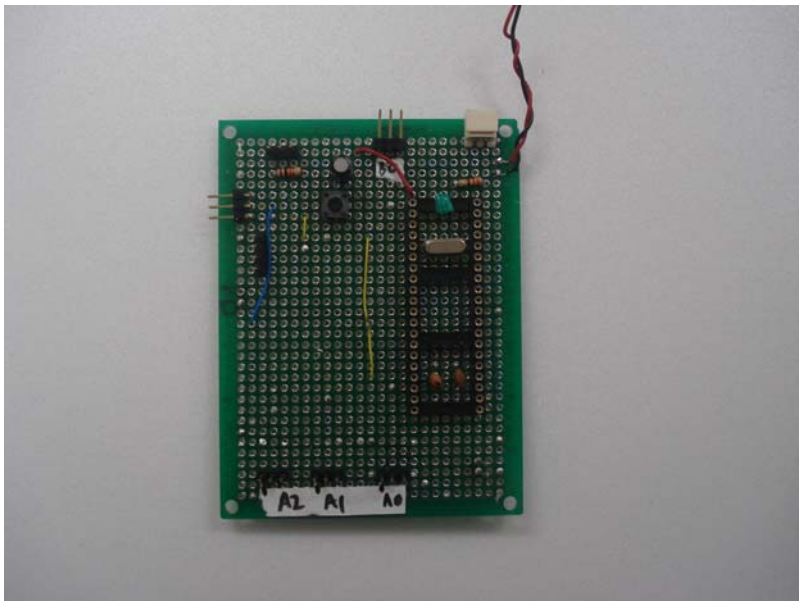
4.1.4. Controller 제작(1차)

Controller는 자동제어에 핵심적인 회로인 만큼 다른 여러 회로를 만들어 회로에 대한 이해를 높인 후 제작했다. 지금까지 사용했던 모든 부품이 종합적으로 들어가게 만들었다.

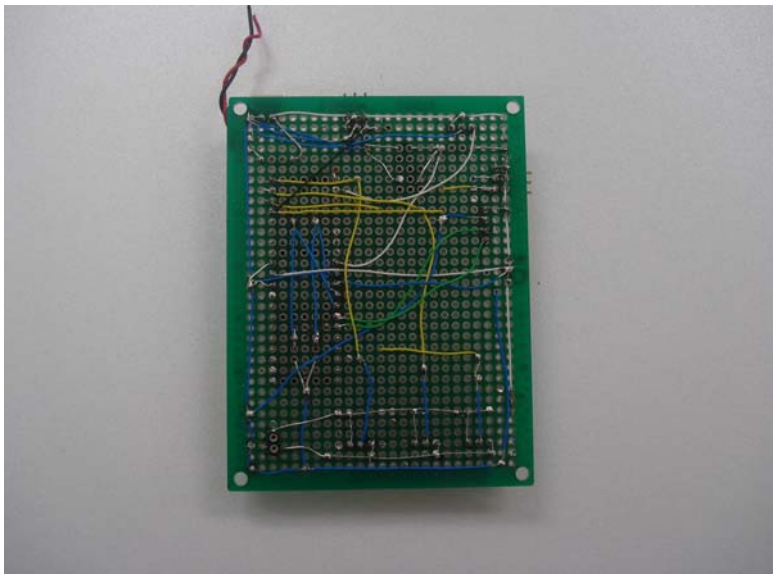
* 준비물 : ATMEGA8535, 30pF 캐퍼시터 2개, 4Mhz 크리스탈, 10μF전해콘덴서, 10KΩ, 저항 3개, 서보모터, 초음파센서 3개, RS232케이블, 전원콘넥터 2개, 커넥터 5개



<그림 61> Controller 제작과정



<그림 62> Controller 앞면

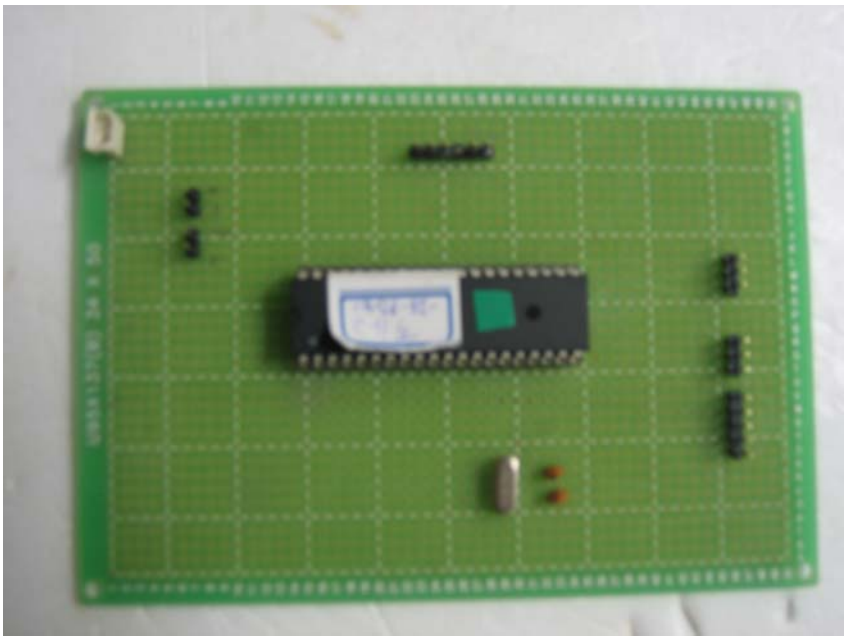


<그림 63> Controller 뒷면

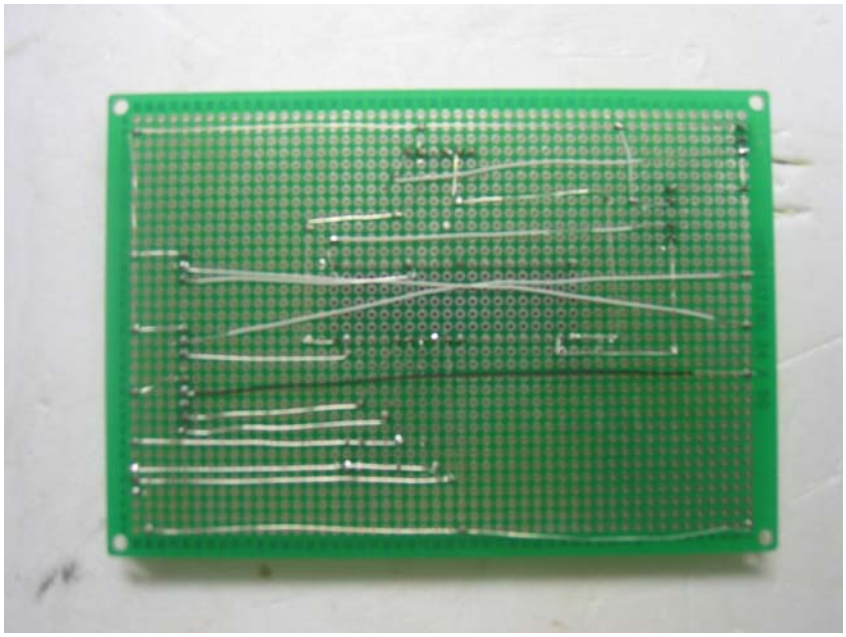
4.1.4. Controller 제작(2차)

* 준비물 ATMEGA8535, 30pF 캐퍼시터 2개, 4Mhz 크리스탈, 서보모터, 초음파센서(SRF04) 2개, RS232케이블, 전원콘넥터 1개, 커넥터 4개

<그림 64> SRF04



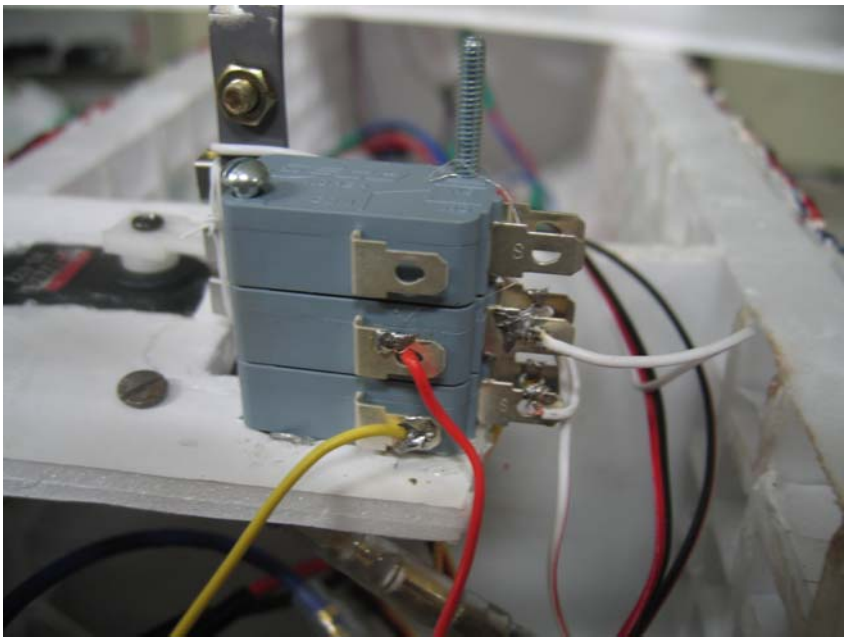
<그림 65> 최종 controller 앞면



<그림 66> 최종 controller 뒷면

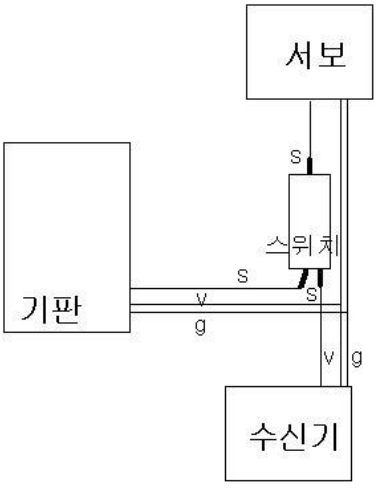
여러 기판을 제작해보며 생긴 문제점들을 보완해서 최종적으로 한 설계에 따라 제작한 기판이다.

자동/수동 변환을 소프트웨어적인 방법을 포기하고 스위치를 써서 변환하게 만들었기 때문에 안정적이고 효율적인 스위치를 만드는 것이 중요하였다.



<그림 67> 스위치

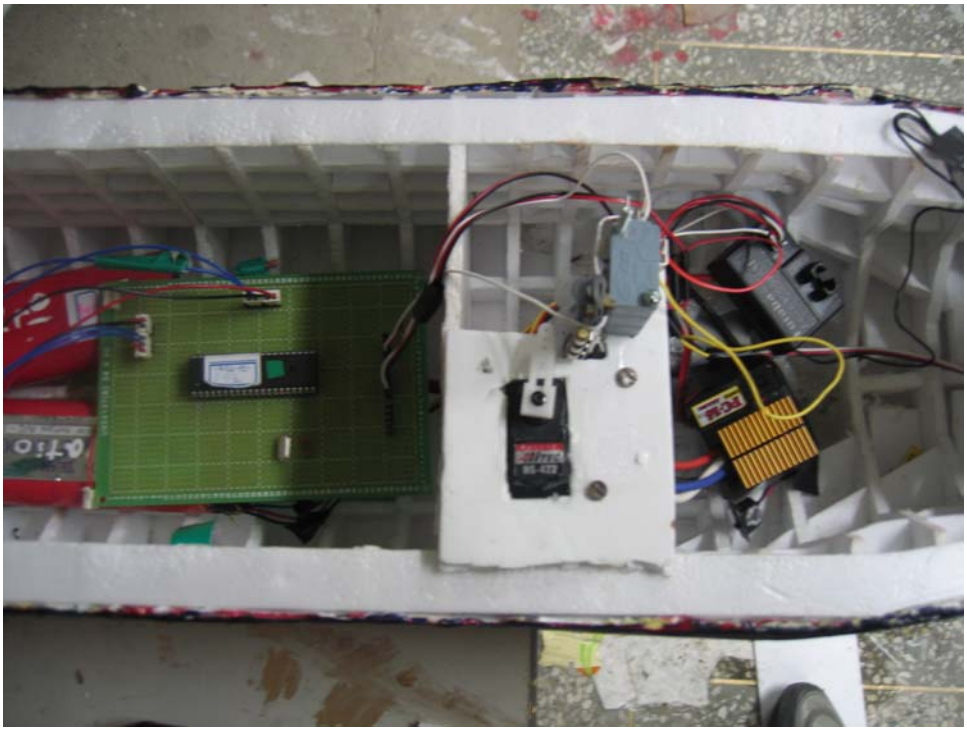
우리조에서는 다른 조와는 달리 스위치를 평면적으로 배치하지 않고 세로로 쌓고 고정하여 공간을 적게 차지하면서도 배선연결이 간단한 스위치를 제작하였다,



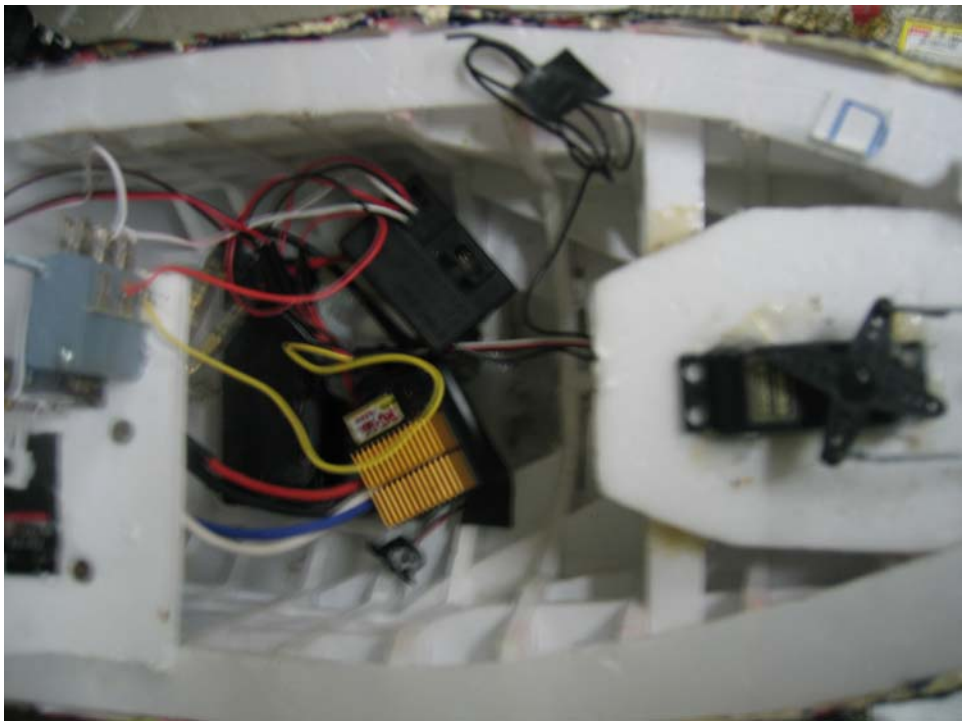
<그림 68> 스위치 개념도

이 스위치가 닫혔을 때는 수신기와 서보모터의 신호선이 연결되고, 열였을

때는 기판과 서보모터의 신호선이 연결된다. 전원과 그라운드는 항상 연결되어 있어도 문제가 없음을 확인 하였다.



<그림 69> 최종적인 배선 모습 1



<그림 70> 최종적인 배선 모습 2

4.2. 소프트웨어

4.2.1. LED 점등 예제 프로그래밍

아래의 LED 점등 예제는 PORTB에 연결된 8개의 led 소자에 먼저 0x01(이진수로는 000000001)을 PORTB로 출력함으로써 0번 비트에 연결된 led를 점등하고 led<<=1 명령을 통해 왼쪽으로 1bit씩 이동시켜 점등 된 led가 순차적으로 왼쪽으로 이동하는 것처럼 보이게 하는 예제이다. 즉 다음과 같이 작동한다.

○○○○○○○●

○○○○○○●○

○○○○○●○○

○○○○●○○○

○○○●○○○○

○○●○○○○○

○●○○○○○○

●○○○○○○○

led에 0x00이 들어가는 가면서 led는 다 소등되는데 이 때 다시 초기상태로 돌아가 무한루프를 돌게 된다..

<표 13> LED 점등 예제 프로그래밍#1

```
#include <mega8535.h>
#include <delay.h>

void main(void)
{
int led=0x01;
DDRB=0xff;
PORTB=led;

while(1)
{
led<<=1;
delay_ms(100);
```

```
if(led==0x00) {led=0x01;}
PORTB=led;
};
}
```

<표 14> LED 점등 예제 프로그래밍#2

```
#include <mega8535.h>
#include <delay.h>
char led_fo[]={0x00, 0x80, 0xc0, 0xe0, 0xf0, 0xf8, 0xfc, 0xfe};
void main(void)
{
DDRB=0xff;
PORTB=0x00;
while(1)
{
int i;
int j;
int led_mov;
for(i=0;i<=7;i++)
{
led_mov=0x01;
```

```
for(j=i;j<=7;j++)
{
delay_ms(100);
PORTB=led_fo[i]+led_mov;
led_mov <<=1;
}
delay_ms(500);
}
for(i=0;i<=2;i++)
{
PORTB=0x00;
delay_ms(500);
PORTB=0xff;
delay_ms(500);
}
```

위 LED 점등 예제 프로그래밍#2는 다음과 같이 동작하게끔 프로그래밍 하였다.

○○○○○○○●

○○○○○○●○

○○○○○●○○

○○○○●○○○

○○○●○○○○

○○●○○○○○

○●○○○○○○

●○○○○○○○

●○○○○○○●

●○○○○○●○

●○○○○●○○

●○○○●○○○

●○○●○○○○

●○●○○○○○

●●○○○○○○

●●○○○○○●

●●○○○○●○

●●○○○●○○

●●○○●○○○

●●○●○○○○

●●●○○○○○

●●●○○○○●

●●●○○○●○

●●●○○●○○

●●●○●○○○

●●●●○○○○

●●●●○○○●

●●●●○○●○

●●●●○●○○

●●●●●○○○

●●●●●○○●

●●●●●○●○

●●●●●●○○

●●●●●●○●

●●●●●●●○

●●●●●●●●

led_fo라는 배열을 선언해서 ●○○○○○○○ ●●○○○○○○ ●●●○○○○○ ●●●●○○○○ ●●●●●○○○ ●●●●●●○○ ●●●●●●●○ 에 해당하는 16진수를 저장한 후 led_mov라는 변수를 선언, 첫번째 예제와 마찬가지로 led를 오른쪽에서부터 한 칸씩 왼쪽으로 이동하게 하는 것과 조합하여 한 칸씩 쌓이는 것과 같이 보이게 프로그래밍 해보았다.

4.2.2. 서보모터 제어 프로그래밍

<표 15> 서보모터 회전 예제 소스 코드

```
#include <mega8535.h>
#include <delay.h>
void main()
{
```

```
int i;
 DDRD = 0x00;


 DDRB = 0xff;
 PORTB = 0x00;




while(1)
 {
   if(PIND.0 == 0)
   {
      for(i=0;i<60;i++)
      {
      PORTB = 0xff;
      delay_us(600);
      PORTB = 0x00;
      delay_us(1400);
      }
   }
```

```
else if(PIND.1 == 0)
{
   for(i=0;i<60;i++)
   {
   PORTB = 0xff;
   delay_us(170);
   PORTB = 0x00;
   delay_us(1830);
   }
}

else
{
   for(i=0;i<60;i++)
   {
   PORTB = 0xff;
   delay_us(370);
   PORTB = 0x00;
   delay_us(1630);
   }
}
```

```
    }
}
```

위는 본격적인 컨트롤러에 탑재할 프로그램의 작성에 앞서 필수 구성 요소라고 할 수 있는 서보모터 제어를 시험해 본 프로그래밍이다. 다른 조의 제어부 팀원과 함께 공동 연구를 진행해본 결과 좌측, 우측, 중앙으로 서보를 회전(좌우측 90도)하게 하는 Pulse Wave의 Width 가 각각 170us, 600us, 370us 임을 여러 차례대의 실험에 의해서 찾을 수 있었다. 위 프로그램은 D 포트의 0번과 1번 bit에 연결된 스위치를 통해서 좌우 회전을 제어한다.

4.2.3. ADC 값 터미널 출력

<표 16> ADC 변환 값 읽기 예제 소스코드

```
#include <mega8535.h>


// Standard Input/Output functions
#include <stdio.h>
#include <delay.h>
unsigned int adc_data;
#define ADC_VREF_TYPE 0x00


// ADC interrupt service routine
interrupt [ADC_INT] void adc_isr(void)
```

```
{
// Read the AD conversion result
adc_data=ADCW;
}


// Read the AD conversion result
// with noise canceling
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input|ADC_VREF_TYPE;
#asm
    in    r30,mcucr
    cbr   r30,__sm_mask
    sbr   r30,__se_bit | __sm_adc_noise_red
    out   mcucr,r30
    sleep
    cbr   r30,__se_bit
    out   mcucr,r30
#endasm
return adc_data;
}
```

```
// Declare your global variables here


void main(void)
{
// Declare your local variables here
int adc_result;


// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTA=0x00;
DDRA=0x00;


// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTB=0x00;
```

```
DDRB=0x00;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTD=0x00;
DDRD=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
```

```
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
```

```
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;
```

```
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: Off
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 9600
UCSRA=0x00;
UCSRB=0x08;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x19;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;
```

```
// ADC initialization
// ADC Clock frequency: 1000.000 kHz
// ADC Voltage Reference: AREF pin
// ADC High Speed Mode: On
// ADC Auto Trigger Source: None
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x8A;
SFIOR&=0xEF;
SFIOR|=0x10;

// Global enable interrupts
#asm("sei")

while (1)
        {
        // Place your code here
           adc_result=read_adc(0);
           printf("ADC_Result = %d ", adc_result);
           delay_ms(10);
        };
}
```

위 소스 코드는 ATMEGA8535에 내장된 ADC 포트를 이용해서 아날로그

값을 디지털 값으로 변환한 수치를 PC를 통해 확인해보는 예제였다. 본 조가 선정한 센서에서 아날로그 값인 전압이 출력되므로 그 값을 ADC를 통해 읽어 들이는 예제의 수행은 필수적이라 판단된다. 프로그램은 상당히 간단하게 작성할 수 있었다. 앞 부분은 대부분 USART, ADC, PORT입출력 설정 파트이고, 실제 코딩은 while 문 안의 구문 뿐이다. ADC 컨버터에서 변환된 값을 읽어 드릴 때는 read_adc 함수를 사용하면 쉽게 해결된다. 그런 다음 콘솔창에 출력하는 printf 명령어만 사용하면 RS-232 케이블과 연결된 PC의 터미널에 원하는 adc_result 값을 출력시킬 수 있다.

4.2.4. Controller 프로그래밍(1차)

```
#include <mega8535.h>


// Standard Input/Output functions
#include <stdio.h>
#include <delay.h>
unsigned int adc_data;
#define ADC_VREF_TYPE 0x00


// ADC interrupt service routine
interrupt [ADC_INT] void adc_isr(void)
{
// Read the AD conversion result
```

```
adc_data=ADCW;
}


// Read the AD conversion result
// with noise canceling
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input|ADC_VREF_TYPE;
#asm
    in    r30,mcucr
    cbr   r30,__sm_mask
    sbr   r30,__se_bit | __sm_adc_noise_red
    out   mcucr,r30
    sleep
    cbr   r30,__se_bit
    out   mcucr,r30
#endasm
return adc_data;
}
int i;
void Servo_left()
{
```

```
for(i=0;i<=30;i++)
{
PORTB.0=1;
PORTB.1=1;
delay_us(170);
PORTB.0=0;
PORTB.1=1;
delay_us(430);
PORTB.0=0;
PORTB.1=0;
delay_us(1500);
}
}
void Servo_right()
{
for(i=0;i<=30;i++)
{
PORTB.0=1;
PORTB.1=1;
delay_us(600);
PORTB.0=0;
PORTB.1=0;
```

```
delay_us(1500);
}
}
void Servo_Init()
{
for(i=0;i<=30;i++)
{
PORTB.0=1;
PORTB.1=1;
delay_us(370);
PORTB.0=0;
PORTB.1=1;
delay_us(230);
PORTB.0=0;
PORTB.1=0;
delay_us(1500);
}
}
// Declare your global variables here
void main(void)
{
// Declare your local variables here
```

```
int sum;
int onoff;
int i;
int sensor1;
int sensor2;
int sensor3;
int receiver;
int ForeDistMin=600;
int SideDistMin=300;
int SideDistMax=600;
// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
```

```
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTD=0x00;
DDRD=0x00;

// Timer/Counter 0 initialization
```

```
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer 1 Stopped
// Mode: Normal top=FFFFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x00;
TCCR1B=0x00;
```

```
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
```

```
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: Off
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud rate: 9600
UCSRA=0x00;
UCSRB=0x08;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x19;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
```

```
ACSR=0x80;
SFIOR=0x00;

// ADC initialization
// ADC Clock frequency: 1000.000 kHz
// ADC Voltage Reference: AREF pin
// ADC High Speed Mode: On
// ADC Auto Trigger Source: None
ADMUX=ADC_VREF_TYPE;
ADCSRA=0x8A;
SFIOR&=0xEF;
SFIOR|=0x10;

// Global enable interrupts
#asm("sei")

while (1)
        {
        sum=0;
        for(i=0;i<19;i++)
        {
        receiver=read_adc(3);
```

```
        sum=sum+ receiver;
        }
        if(sum>=100)
         {onoff=0;}
         else
         {onoff=1;}
        while(onoff)
        {
          sensor1=read_adc(0);
          sensor2=read_adc(1);
          sensor3=read_adc(2);
          if((sensor1>=0) && (sensor1<=ForeDistMin) || (sensor2>=0)
&& (sensor2<=SideDistMin))
          {Servo_left();}
          else
          {
          if(sensor2>=SideDistMax)
          {Servo_right();}
          else
          {Servo_Init();}

        }};
```

```
    };
}
```

초음파 센서를 정상적으로 구동시키지 못한 관계로 컨트롤러 전체를 통괄하는 위 프로그램을 시험해보지 못했다. 앞서 소개한 알고리듬과 동일하며 PWM으로 서보 모터와 변속기를 제어하게 된다. 이 중 특이할만한 것은 무인제어에서 수동제어 및 수동제어에서 무인제어로 전환하는 과정이다. 이 과정을 통상적으로 고안되었던 2 종류의 크리스탈과 2개의 수신기를 이용해 MPU의 전원을 차단시키는 방법이 아닌 단 하나의 수신기를 사용하여 그 신호를 제어할 수 있었다는 점이다. 상당히 간단한 방법으로 송신기의 전원 스위치를 키게 되면 일단 Pulse파가 수신기를 통해서 들어오게 되고 그 수신기의 신호를 adc 를 통해 읽어 들이는 것이다. 송신기가 꺼져있는 경우, read_adc()함수를 통해 수신기의 펄스파를 읽어 들인 결과 당연하겠지만 0 값들이 반복됨을 알 수 있었고, 켜져 있는 경우에는 0이 반복되다가 주기적으로 800이상의 값이 검출됨을 알 수 있었다.

4.2.5. Controller 프로그래밍(2차)

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.25.2 Evaluation
Automatic Program Generator
?Copyright 1998-2006 Pavel Haiduc, HP InfoTech s.r.l.
```

```
http://www.hpinfotech.com

Project :
Version :
Date    : 2006-11-18
Author  : Freeware, for evaluation and non-commercial use only
Company :
Comments:


Chip type               : ATmega8535
Program type            : Application
Clock frequency         : 4.000000 MHz
Memory model            : Small
External SRAM size      : 0
Data Stack size         : 128
*****************************************************/

#include <mega8535.h>
#include <delay.h>

// Standard Input/Output functions
```

```
#include <stdio.h>


// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
TCNT0=0x16;
PORTD.6=1;
PORTD.7=1;
delay_us(10);
PORTD.6=0;
PORTD.7=0; // 오버 플로우가 발생할 때마다 D포트의 6,7 bit에 폭이 10uS
인 trigger pulse를 발생시키는 부분이다.


}


unsigned int adc_data;
#define ADC_VREF_TYPE 0x00


// ADC interrupt service routine
interrupt [ADC_INT] void adc_isr(void)
{
// Read the AD conversion result
```

```
adc_data=ADCW;
}

// Read the AD conversion result
// with noise canceling
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
#asm
    in    r30,mcucr
    cbr   r30,__sm_mask
    sbr   r30,__se_bit | __sm_adc_noise_red
    out   mcucr,r30
    sleep
    cbr   r30,__se_bit
    out   mcucr,r30
#endasm
return adc_data;
}

// Declare your global variables here
int i;
```

```
int fo_dist_min=200;

int si_dist_max=250;

int si_dist_min=150;

int fo_dist;

int si_dist;


int count;


void rotate(void)

{

            for(i=0;i<=30;i+ + )

          {

            PORTC.5=1;

           delay_us(350);

           PORTC.5=0;

           delay_ms(10);

           PORTD.6=1;

           delay_us(400);

           delay_ms(10);

          }

}
```

```
void to_left(void)
{
        for(i=0;i<=30;i++)
        {
         PORTC.5=1;
         delay_us(350);
         PORTC.5=0;
         delay_ms(10);
        }
}

void to_right(void)
{
         for(i=0;i<=30;i++)
        {
         PORTC.5=1;
         delay_us(490);
         PORTC.5=0;
         delay_ms(10);
        }
}
void center(void)
```

```
{
          for(i=0;i<=30;i++)
         {
          PORTC.5=1;
          delay_us(420);
          PORTC.5=0;
          delay_ms(10);

         }
}
int get_fo_dist(void)
{
        count=0;
        for(i=0;i<=2000;i++)
        {
        if(PINB.0==1)
        {
        count=count+1;
        }
        delay_us(10);
        }
        return count;
```

```
}


int get_si_dist(void)
{
        count=0;
        for(i=0;i<=2000;i++)
        {
        if(PINB.1==1)
        {
        count=count+1;
        }
        delay_us(10);
        }
        return count;
}


void main(void)
{
// Declare your local variables here


// Input/Output Ports initialization
// Port A initialization
```

```
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTB=0x00;
DDRB=0x00;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTC=0x00;
DDRC=0xff;
```

```
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In
Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T
State0=T
PORTD=0x00;
DDRD=0xff;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 15.625 kHz
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x04;
TCNT0=0x16;
OCR0=0x00; // 주기가 60ms인 overflow 인터럽트를 발생시키기 위한
Timer0의 설정부분이다.

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 15.625 kHz
```

```
// Mode: Fast PWM top=00FFh
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x81;
TCCR1B=0x0C;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1A=5;
OCR1BH=0x00;
OCR1BL=0x00; //Timer1 을 통해 DC motor 변속기를 제어할 pwm을 생성
하는 부분이다.

// Timer/Counter 2 initialization
// Clock source: System Clock
```

```
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: Off
// USART Transmitter: On
```

```
// USART Mode: Asynchronous
// USART Baud rate: 19200
UCSRA=0x00;
UCSRB=0x08;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x0C;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// ADC initialization
// ADC Clock frequency: 1000.000 kHz
// ADC Voltage Reference: AREF pin
// ADC High Speed Mode: On
// ADC Auto Trigger Source: None
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x8A;
SFIOR&=0xEF;
```

```
SFIOR|=0x10;


// Global enable interrupts
#asm("sei")


while (1)
        {
        count=0;
        for(i=0;i<=2000;i++)
        {
        if(PINB.0==1)
        {
        count=count+1;
        }
        delay_us(10);
        }
        fo_dist=count;
        printf(" SENSOR 1 %5d   ", fo_dist);



        count=0;
        for(i=0;i<=2000;i++)
```

```
        {
        if(PINB.1==1)
        {
        count=count+ 1;
        }
        delay_us(10);
        }
        si_dist=count;
        printf(" SENSOR 2 %5d ", si_dist);

        if(fo_dist<=fo_dist_min)
        {to_left();}
        else if(si_dist<=si_dist_min)
        {to_left();}
        else if(si_dist>si_dist_max)
        {to_right();}
        else
        {center();}
        };
}
```

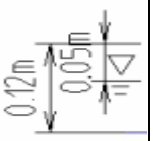
1차 Controller Program에 비해서 달라진 점은 크게 두 부분이다. 1차의 경우 506nv 를 사용해 별다른 설정 없이 ADC 를 통해 변환된 값만 읽어 들이면 센서의 출력 값을 얻을 수 있었던 것과는 달리 따로 trigger pulse

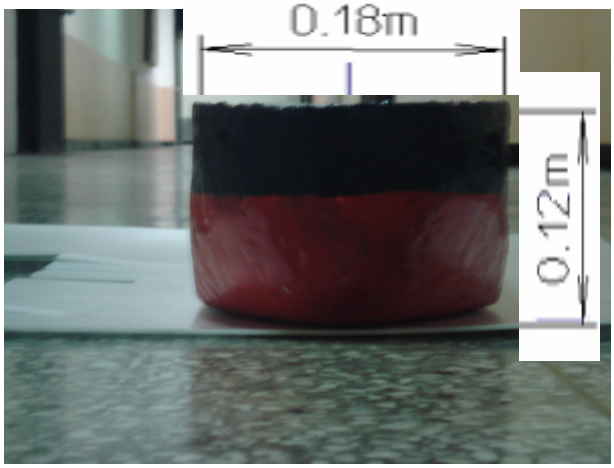
를 입력해주는 부분이 추가 된 것이 그 중 하나이다. 또 하나는 DC motor의 제어가 서보 모터와 같이 pulse 파의 생성을 port 출력을 통해 생성한 것이 아니라 내부 타이머 기능을 이용하여 pwm 파형을 생성하여 DC motor의 rpm을 일정하게 유지하도록 한 점이다.

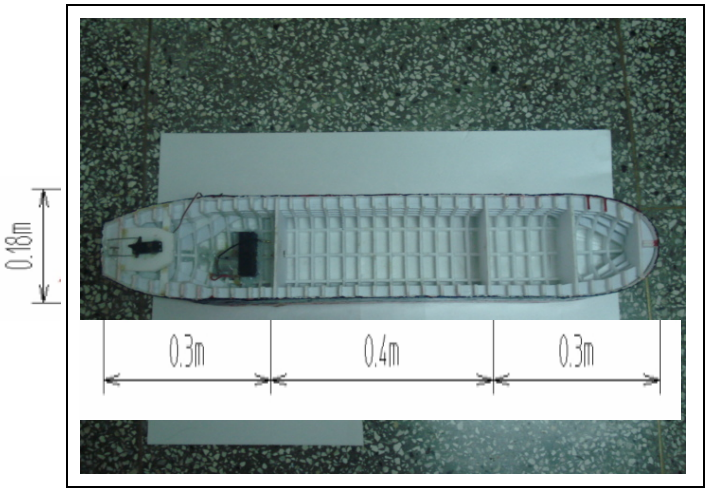
5. Contest

5.1. 완성된 모형선의 모습 및 시운전

5.1.1. 완성된 모형선의 개략배치도

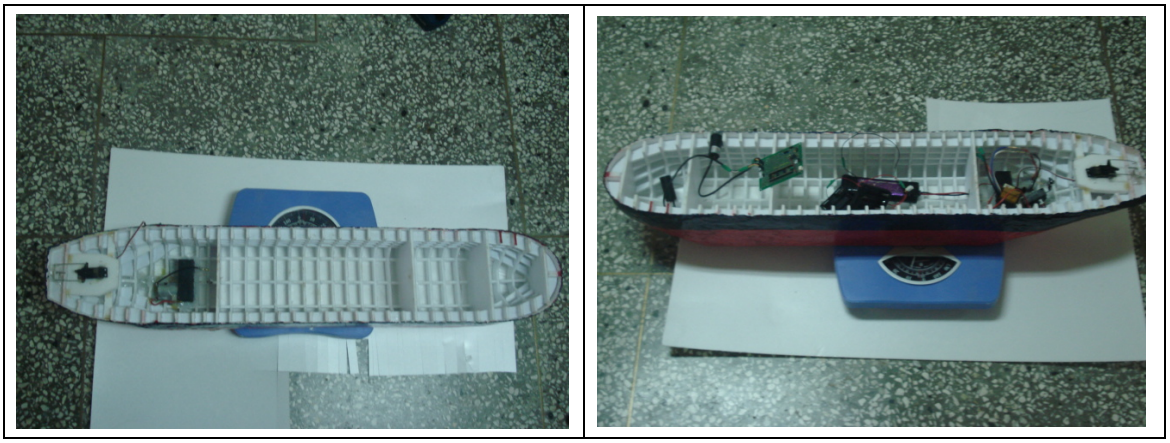






<그림 71> 완성된 모형선의 개략배치도

5.1.2. 완성된 모형선의 개략배치도



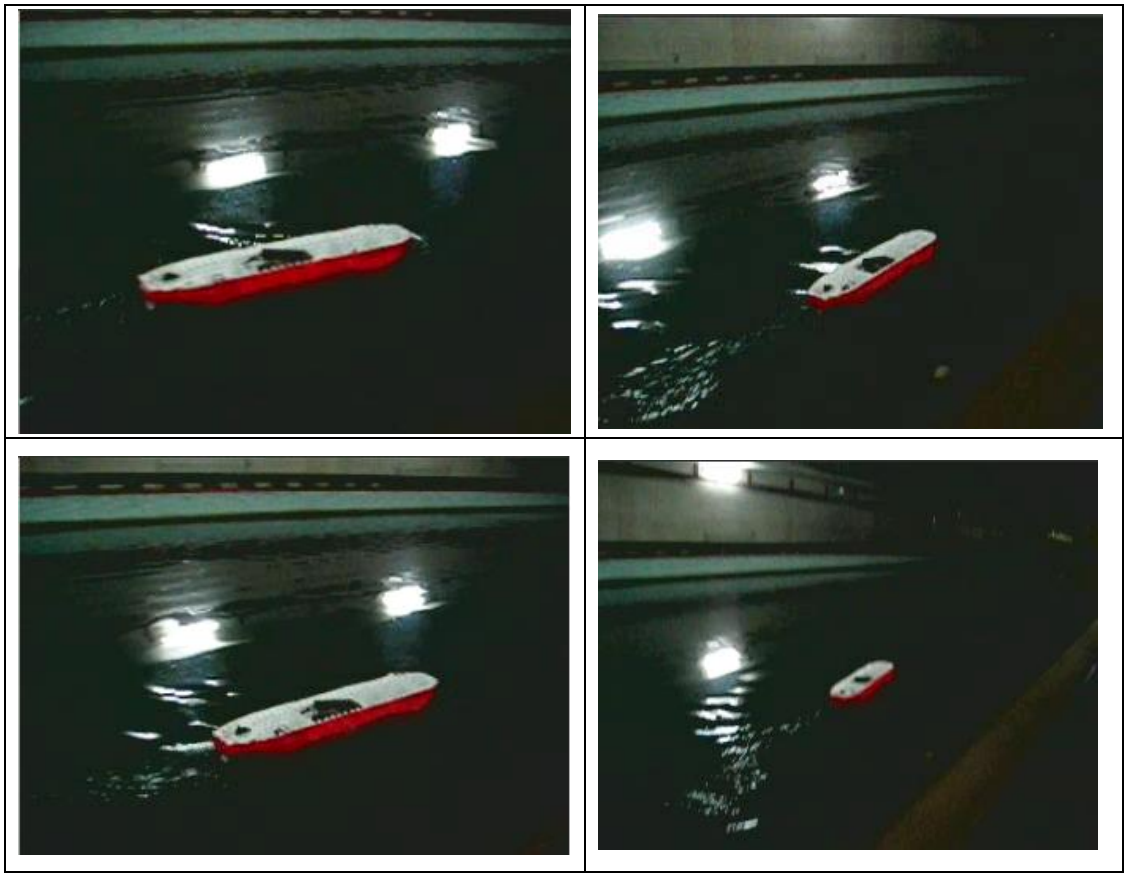
<그림 72> 경하중량 실측

<표 17> 경하중량

선체	기관부			추진부	계
2.5(kg)	배터리	모터	센서 및 기타회로	0.05(kg)	1(kg)
	0.5(kg)	0.05(kg)	0.4(kg)		

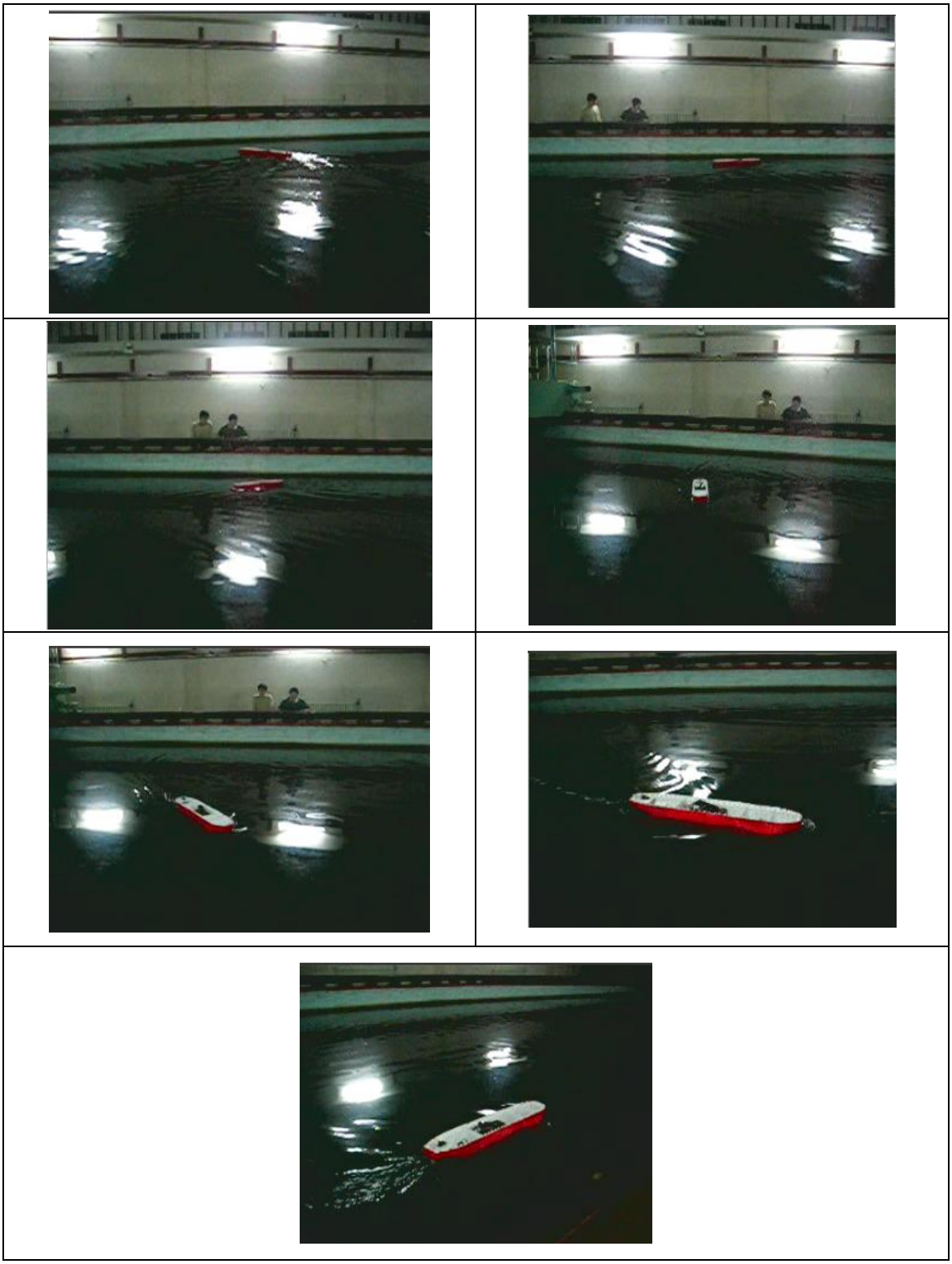
5.1.3. 주행능력 시험

5.1.3.1.직진성 확인



<그림 73> 직진성 확인

5.1.3.2. 선회성 확인



<그림 74> 선회성 확인

5.2. 1차 Contest

5.2.1. 1차 컨테스트 준비

5.2.1.1. 선체부

11월 7일 우리조는 자동제어 부분과 홀수선 긋는 것을 뺀 선체 모든 부분을 완성했다. 선체 자체는 일찍 완성했으나 추진부를 만드는데 시간이 오래 걸려 완성이 늦어졌다. 제작법도 모르는 상태에서 부품들을 조립했고 처음 다뤄보는 변속기 때문에 수동조종이 쉽지 않았다. 특히 다른 조는 다 되는 변속기가 계속 안되 고생했는데 원인을 알고 보니 단순한 접촉 문제여서 매우 허탈했다.

컨테스트 하기 전 우리가 만든 배가 실제로 수면에서 잘 작동하는지 확인해 보려고 수조로 시운전을 하러 갔다. 처음 배를 물에 띄우고 조종기를 통해 제어를 했을 때의 그 기쁨은 지금도 생생히 기억난다. 기쁜 마음으로 배를 물에서 꺼내서 보니 좌측 중앙평행부 바닥에 물이 조금 고여있는 것을 발견했다. 당연히 선체자체는 방수처리를 잘 했다고 생각하고 작년 선배들의 조언을 듣고 샤프트관과 그 주위 방수 처리에 세심히 신경을 썼는데 선체가 물이 새버렸다. 방수처리가 제대로 된지 않는다는 것은 컨테스트를 하는데 있어 매우 큰 결함이었다. 자동제어를 생각하기 이전에 기본이 안되어있는 것을 뜻한다. 대책회의 끝에 퍼티작업부터 다시 시작해서 선체 외판을 단단하게 하기로 했다. 그 이유로는 처음에 1차 퍼티작업을 할 때 경험부족으로 선체외형이 매끄럽지 못했고, 선체 무게 자체도 처음 설계했을 때의 값과 많이 차이 나고, 선체 자체 방수처리를 더 꼼꼼하게 하기 위해서였다. 하지만

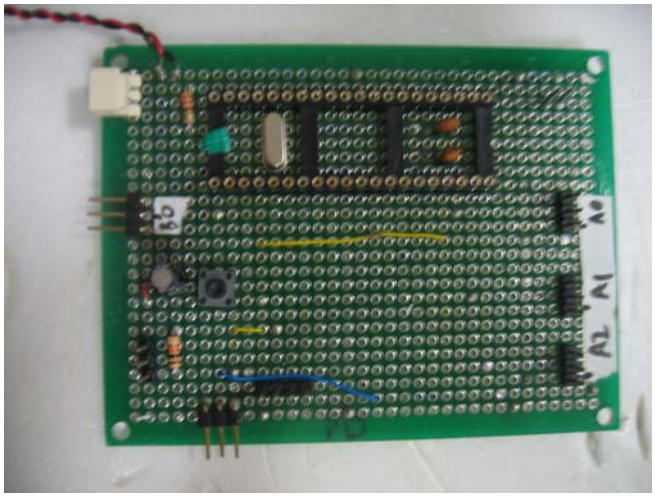
11월 8일 최종보고서 발표날에 발표할 때 배를 보여주고 수동제어 확인을 받아야 하기 때문에 바로 퍼티작업에 들어갈 수 없어 발표 끝나고부터 작업에 들어가기로 했다. 11월 8일 발표를 위해 시운전 때 지정된 재화중량을 싣고 생기는 흘수선을 찾아 저녁에 페인트 칠을 하였다. 칠한 후 말리는 작업 중 부주의로 인해 테이블에서 배를 떨어트렸다. 떨어졌을 때의 충격으로 선수와 중앙평행부의 연결부분 한쪽 면에 한 줄 금이 갔다. 다행히도 외판을 이루는 우드락까지 금이 간 것이 아니라 퍼티가 굳은 부분이 외판과 떨어져 금이 갔다. 하지만 이 상태로는 컨테스트가 불가능하다. 방수처리의 전면 재수정함에 또 하나의 이유가 생긴 샘이다.

11월 8일 최종보고서 발표가 끝나자마자 바로 퍼티작업에 들어갔다. 처음에 실패한 것을 경험삼아 좀더 세심하게 골고루 펴 발랐다. 홈이 생기거나 너무 많이 퍼티가 발리는 것에 조심하였다. 11월 11일 컨테스트 전까지 시간이 너무 부족하였기에 퍼티작업은 1차로 끝내고 11월 9일에 바로 에폭시 작업을 하였다. 오전에 일찍 에폭시를 발랐더니 저녁때쯤 말라 바로 페인트 칠을 할 수 있었다.

11월 10일 오전에 나머지 페인트 칠, 즉 흘수선을 그었다. 방수처리작업을 다시 했더니 어느정도 무게가 늘어나 적당한 높이로 흘수선을 그을 수 있었다. 여기서 적당한 높이란 너무 낮지도 높지도 않은 높이를 말한다. 흘수선이 너무 높으면 건현이 낮아져 운항중 주변 파도에 의해 침수 위험이 있게 되고, 너무 낮으면 복원력이 작아지면 rolling에 약해져 침수 위험이 있기 때문이다.

5.2.1.2. 제어부

11월 3일 자동제어를 위한 제어기판을 최초로 제작하였다. 기판은 과제를 수행할 때 서보모터를 제어하기 위해 만들었던 기판에 약간의 수정을 해서 만들었다.



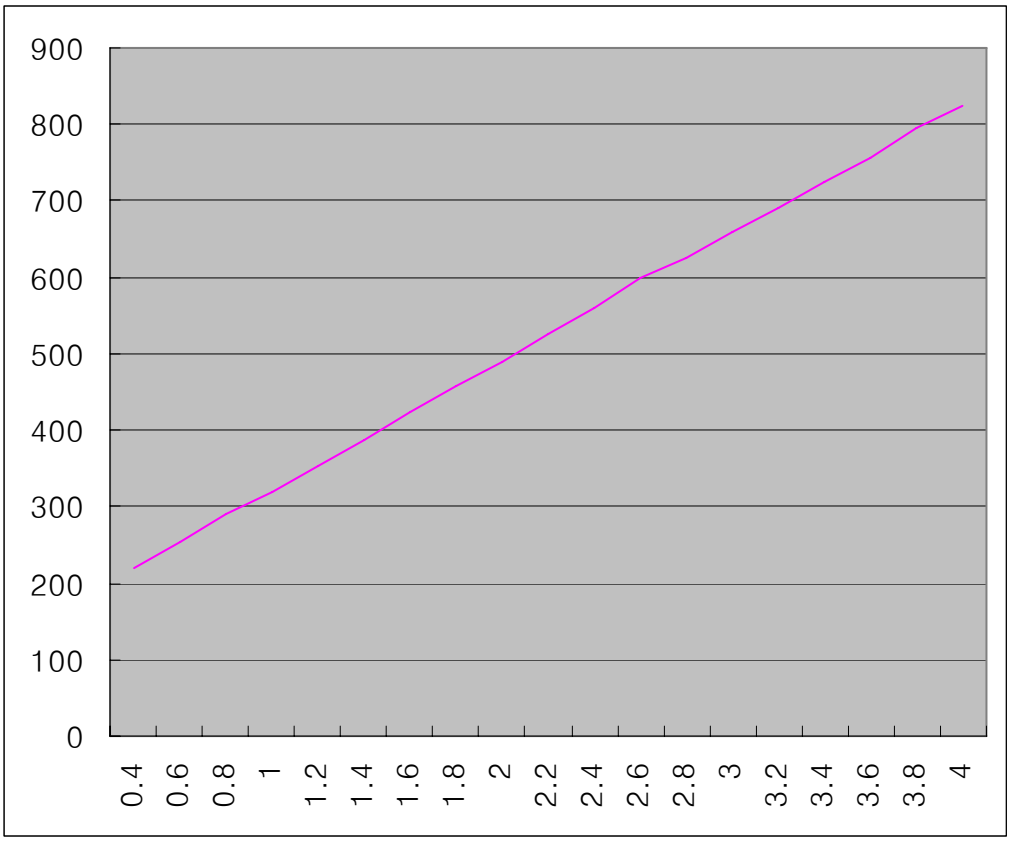
<그림 75> 자동제어기판a

기판에는 센서에 입력값을 받기 위해 3개의 커넥터가 포트A0, A1, A2에 연결되어 있고 서보모터와 변속기를 제어하기 위한 커넥터를 설치했다.

11월 7일 작성된 기판으로 소프트웨어를 작성하려고 하니 기판에 문제가 있었다. 20pin커넥터에 선연결이 제대로 되지 않았다. 그래서 전선을 수정했다. 수정된 기판을 가지고 센서를 연결하여 RS232를 통해 컴퓨터로 입력 값이 표시되는 간단한 프로그램을 만들어서 센서 입력값을 테스트 해보았다. 그러나 값이 완전 랜덤하게 나왔다. 센서에 12V의 전압만 걸어주면 바로 작동이 될 것이라는 예상을 완전히 벗어난 것 이었다. 문제의 원인이 기판에는 5V의 전압이 흐르고 AVR에서 사용하는 레퍼런스 전압도 5V를 걸어주어 센서에 들어가는 12V와 레퍼런스전압이 달라서 발생한 문제라고 생각하고

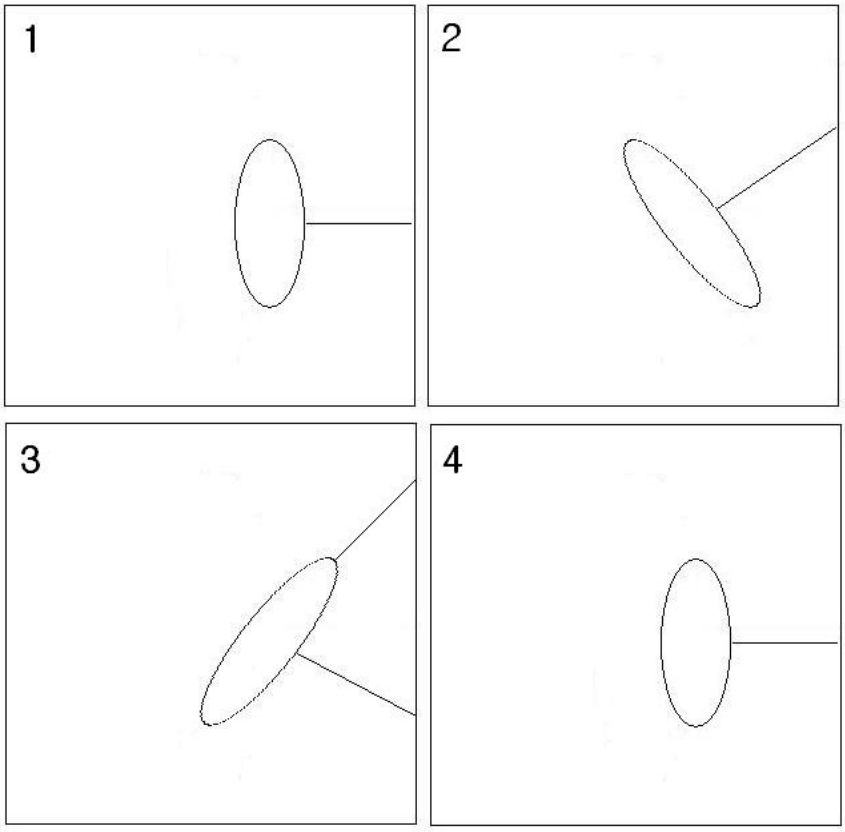
AVR의 레퍼런스 전압을 12V로 연결해 주었다가 같이 실험을 해보던 2조의 기판이 타 버렸다. 상황을 해결하기위해 우리조에서는 센서회사인 센서텍에 연락을 해서 기술자와 통화를 하였으나 센서의 사용법에 대해선 아무것도 알아 낼 수 없었다. 우리는 센서가 불량일 확률이 높다고 생각하고 더 이상 작업을 진행하지 못했다.

11월 9일 이기영군과 김명수군이 대표로 센서텍회사에 직접 갔다. 우리조에서는 이들이 새로운 점을 알아오면 바로 새로운 기판을 작성할 수 있도록 용산에가서 부품을 넉넉히 구매해 놓았다. 저녁에 이기영군이 돌아왔으나 센서에는 아무 문제가 없다는 대답만을 들었다고 했다. 그렇지만 12V의 전압을 레귤레이터를 이용해 5V로 낮추어 사용해보자고 했다, 센서에 12V, 기판에 5V의 전압을 따로 연결하는 것이 아니라 센서에는 12V를 공급해주고 기판에는 레귤레이터를 이용하여 전압을 5V로 떨어뜨려서 공급하자는 것 이었다. 반신반의 하는 마음으로 실험을 해보았다. 다행히도 같은 전압을 떨어뜨려 사용하니 센서가 어느정도 의미있는 값을 출력하는 것을 알 수 있었다.



<그림 76> 거리에 따른 센서값

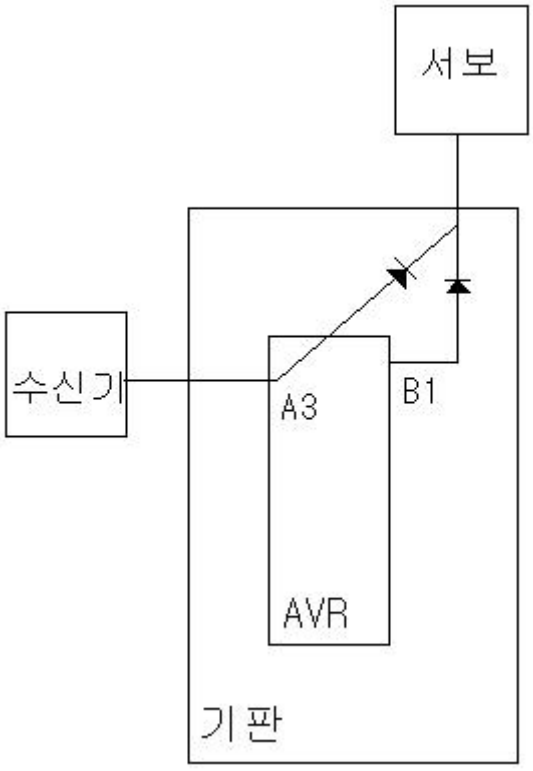
그러나 센서를 테스트 해본 결과 센서와 벽의 각도가 직각일 때에는 센서값과 거리가 위와 같은 비례적인 값을 가졌으나 센서와 벽의 각도가 조금만 틀어져도 거리에 훨씬 큰 값이 나오게 되는 것을 알 수 있었다. 즉 직각이 아닐 경우에는 센서에서 실제의 거리보다 훨씬 멀리 떨어져 있다고 인식한다는 것이다. 그래서 알고리즘을 오른쪽 센서가 너무 멀어지면 러더를 오른쪽으로 틀어서 벽에 가까워지려 하고, 앞쪽 센서에 벽이 감지되면 바로 러더를 왼쪽으로 틀어서 벽과 거리를 늘리는 알고리즘이면 이러한 센서로도 충분히 벽을 타고 진행 할 수 있다고 판단 하여 레귤레이터를 달고 센서값을 읽을 수 있는 제어를 위한 기판을 새로 제작했다.



<그림 77> 초음파센서의 단점을 보완하기위한 알고리즘

11월 10일 이제 남은 일은 자동/수동 변환만 하면 된다고 생각하고 자동/수동 변환에 힘을 쏟았다. 우리조는 다른조와는 달리 자동/수동 전환을 소프트웨어적인 방법으로 하기로 했다. 수신기와 기판을 연결해주어 수동제어를 위한 리모콘이 켜졌을 때 수신기의 펄스값이 바뀌는 것을 이용해서 기판에서 수신기의 바뀐 필스파를 받아들이면 자동제어를 중단하고 수동으로 전환되는 시스템 이었다. 이렇게 할 경우 자동/수동 전환을 위해 번거롭게 여러 개의 송신기나 크리스털을 들고 다닐 필요가 없이 송신기에 전원만 열결하면 자동으로 자동제어가 멈추기 때문에 훨씬 편리하다. 또 스위치 등이 필요 없기 때문에 배를 훨씬 가볍게 만들 수 있고 스위치를 켜고 끄는 데서

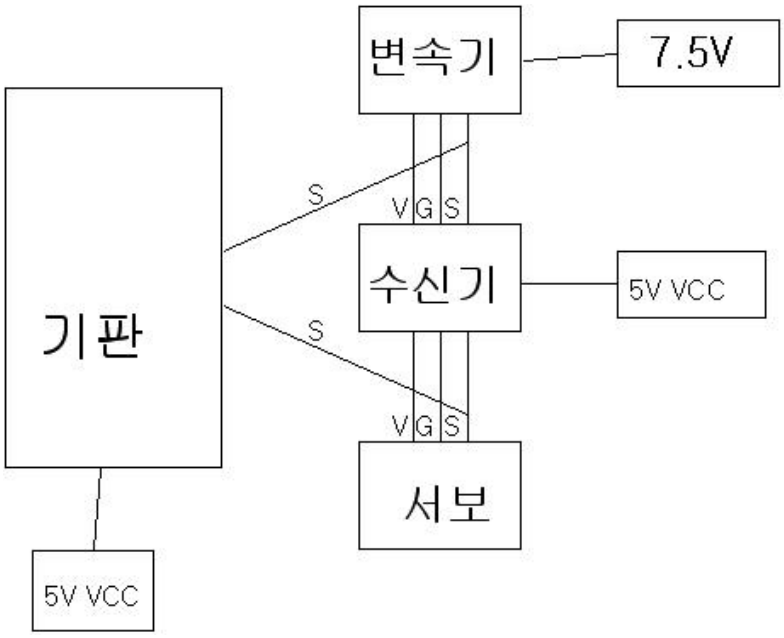
발생할 수 있는 물리적인 문제도 피할 수 있다. 우리조에서는 송신기가 켜졌을 때 수신기에서 바뀐 펄스값을 AVR이 인식한다는 것을 RS232를 통해서 확인하는데 까지 성공하였다. 이제 서보모터와 변속기를 제어하는 기판과 송신기가 켜지는 것을 인식하는 기판을 하나의 기판에 합치는 일만 하면 됐다. 그러나 두 가지 기능이 각각 동작하는 기판은 만들었으나 두 기능이 한꺼번에 되는 기판을 만들어서 RS232를 통해 테스트를 해보니 서보모터나 변속기가 커넥터에 연결되었을 때는 송신기가 켜져 있는지를 확인할 수 없었다. 이것은 수신기에서 들어오는 신호가 역류했을 것이라고 판단, 신호의 역류를 막을수 있는 다이오드를 달았다.



<그림 78> 다이오드를 설치한 기판

이렇게 해서 신호의 역류를 막았으나 여전히 작동이 안 되었다. 심지어 이

렇게 연결하자 원래 작동되던 서보모터, 변속기의 제어가 다시 안 되는 결과가 생겼다. 이것은 다이오드가 신호의 역류를 막아주기는 하지만 기판 전체의 전압을 바꾸어 버려서 AVR과 서보모터에 들어가는 전압이 달라지는 것으로 추정된다. 여러 번의 시도 끝에 전압과 전류의 변화가 어떻게 되는지 전기적인 지식이 부족해 도저히 알 수 없었다. 결국 소프트웨어를 통한 자동/수동 제어 변환을 포기하고 스위치를 통해 변환하기로 했다. 그것을 위해 기판에 변환기능을 없앤 새로운 기판을 다시 제작하였다. 이 기판에는 서보모터와 변속기에 연결되는 커넥터에 전원, 그라운드, 신호선이 모두 연결되어 있었다. 그러나 변속기와 수신기, 서보모터와 수신기를 연결하는 선에서 신호선만을 기판에 연결해 주었다. 나중에 알게 된 것이지만 이것 때문에 계속 자동제어가 안 되는 원인이 되었다. 결국 센서는 차치하고서라도 기판을 통한 변속기와 서보모터의 제어라는 과제도 해결하지 못 한체 컨테스트에 나갈수 밖에 없었다.



<그림 79> 최초의 제어부 연결

5.2.2. 1차 컨테스트 결과 및 향후 과제

11월 11일 자동제어가 제대로 되지 않는 상태에서 컨테스트에 나갔다. 자동제어가 되지 않아 걱정했지만 다행히 그날은 수동제어만 테스트 한다고 했다. 다행히 우리조는 컨테스트 전날 직진성을 좋게 하기위해 러더의 위치를 정중앙에 맞추고 리모컨을 통한 수동제어를 2시간 동안 연습을 했었다. 한가지 걱정되는 점은 아침에 충전시켜둔 베터리를 잃어버려서 전날 저녁부터 연습하고 테스트할 때 썻던 베터리를 그대로 사용하여 운전도중 베터리가 다 닳지 않을까 하는 것이었다. 컨테스트날 아침이 되면 모든조가 하나도 없기 때문에 베터리를 잘 챙기지 않으면 큰 낭패를 본다. 다행히 컨테스트가 끝날 때 까지는 배터리가 남아서 운행을 무사히 마칠 수 있었다. 컨테스트 결과 우리배가 직진성에서 상당히 우수한 면을 보이며 1등을 했다. 뛰어난 직진성을 보일 수 있었던 점은 먼저 우리배가 다른 배들에 비해 유선형으로 생겼기 때문일 것이다. 또 전날 연습을 해보며 러더를 중앙에 정확하게 맞추는 작업도 중요했다. 약간이라도 러더가 틀어져 있으면 방향이 상당히 크게 바뀌기 때문에 러더 맞추는 일이 매우 중요하다. 또 다른 배들에 비해 우리배위 프로펠러와 러더의 간격이 좁았던 점이 러더의 효과를 극대화 시켜 준 것 같다. 우리배에서 나타난 문제점은 6KG의 재하중량을 실었을 경우 흘수가 흘수선과 약간 맞지 않았다. 이것은 흘수선을 그을 때 제어부의 배치와 추의 놓는 위치가 컨테스트 날 바뀌어서 뒤쪽으로 배가 약간 기울면서 앞쪽의 흘수선이 맞지 않게 되었다.

이 날은 수동제어에서 우리배가 좋은 성능을 보여주기는 했지만 자동제어

가 전혀 안 되고 센서의 문제가 하나도 해결되지 않아서 2차 컨테스트 때까지 일주일의 시간동안 이 문제를 해결해야 되는 문제를 남겼다.

5.3. 2차 Contest

5.3.1. 2차 Contest 준비과정

1차 Contest에서 우수한 직진 안정성을 검증 받은 덕분에 추가적인 선체부의 보완 작업은 불필요하다고 판단되었다. 몇 개 조를 제외한 대부분의 조가 직진 안정성을 확보하지 못해 추가적으로 skeg 설치 혹은, 러더의 크기 및 프로펠러와의 간격 수정 등을 통해 보완 작업이 필요했던 것에 비하면 본 조는 보다 많은 시간과 역량을 제어부에 집중할 수 있었다.

1차 Contest 준비로 인한 피로를 주말 동안의 휴식을 통해 푼 다음에 본격적으로 월요일부터 2차 Contest 준비를 시작했다. 해결해야 할 과제들을 정리해보자면 다음과 같았다.

- 센서의 안정적인 구동
- 안정적 자동/수동 전환 방법 확보
- 자동 모드 시 Rudder와 DC motor 의 S/W 제어

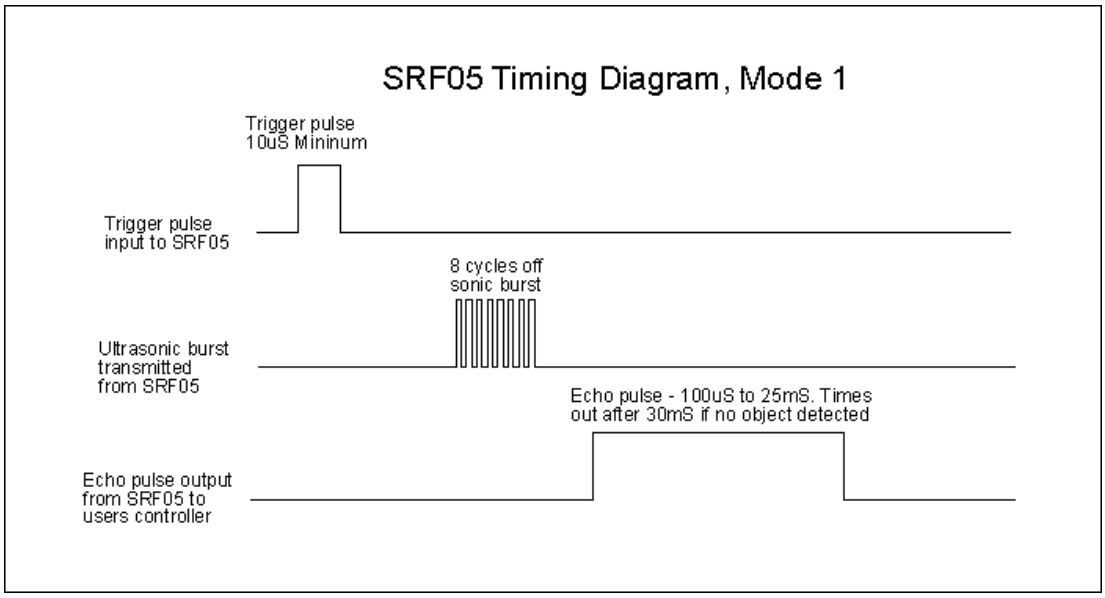
이 문제들 중 어느 하나라도 확실하게 해결하지 못한다면 2차 Contest의 주안점이라고 볼 수 있는 무인 제어는 실패로 돌아갈 것임이 확실했다.

1차 Contest 당시 사용하고 있던 506nv 초음파 센서를 변경하기에 앞서 다시 한번 정확히 상태를 체킹하기 위해 calibration 작업을 다시 수행 하였

다. 지난 주 calibration 때 센서에 대해 수직인 벽면에 대해서는 상당히 만족할만한 분해능을 보여주었기 때문에 문제점들을 보완한다면 충분히 사용가능하리라 판단했다. 따라서 적외선 센서나 타 종의 초음파 센서로 교체하는 것은 잠시 유보해두고 기존의 506nv 모델을 간단한 시험용 기판에 연결하여 터미널로 출력 값을 다시금 검토하였다. 화요일이 지나가도록 문제는 여전히 발견되었고 그 해답이 보이지 않았다. 센서의 검출면이 수직에서 5도 이상만 비틀어져도 출력 값이 크게 변해 사용할 수 없을 정도의 수치가 나와 버렸고 진동 등의 움직임이 있을 때는 물체가 있음에도 몇 초간 미검출 상태가 지속되면서 1v의 출력이 나와 그 값의 처리가 상당히 곤란했다. 실제 배의 움직임을 예상해볼 때 센서와 벽면이 수직을 이루면서 진행하는 경우는 거의 없을 것이므로 더욱 기존 센서의 활용이 힘든 상황이었다. 그 와중에도 김기영군이 조장으로 있는 6조는 그 단점을 보완하기 위해 5개의 센서를 이용함으로써 다수의 센서들 중 하나는 항상 벽면에 수직에 가까운 각도를 유지하게끔 하는 아이디어를 내기도 였다. 하지만 여전히 센서의 오작동으로 인한 출력값을 처리하기란 상당히 어려운 작업이었고 안정성 없는 출력값을 가지고서는 알고리즘을 구성하기가 힘들었다.

수요일, 2차 Contest를 3일 남겨둔 시점에서 우리 조는 센서를 변경할 수 밖에 없다는 판단을 내렸고 그 즉시 대체 가능한 타 센서의 조사를 시작했다. 그 결과 적외선 센서와 srf04 라는 초음파 센서가 고려되었는데 적외선 센서는 측정 거리가 1.5m가 채 되지 않아 후보에서 제외시켰다. 새로 선택한 srf04 라는 초음파 센서는 미니마우스나 여러 로봇에 흔하게 쓰이는 센서이기 때문에 본 과제에 적합할 것이라 생각했다. 일단 한번 시도해보자 하는 생각으로 센서를 구매한 후 그 작동 시험을 시작하였다.

센서를 교체한 후에는 그 작동에 어려움을 겪었다. 기존의 506nv 센서가 12V전원을 인가하기만 하면 별다른 조작이 필요없이 1~5v를 출력했던것에 비해 새로 구입한 srf04 초음파 센서 모듈은 아래 그림에서 보는 것과 같이 10us 폭의 Trigger pulse를 인가함으로써 모듈에서 초음파를 발생시켜 그 측정이 이루어지는 방식이었다.



<그림 80> SRF04 작동 원리

측정이 실시간으로 이루어져야 하고 그 Echo pulse 의 폭에 비례해서 거리가 출력되므로 주기적으로 Trigger pulse를 발생시키는 방법과 Echo pulse의 폭을 측정하는 방법의 연구가 필수적이었다. 시간은 촉박했으나 다른 대안이 없었기 때문에 초심으로 돌아가서 다시 AVR 관련 책들을 참조하면서 배경 이론 공부에 들어갔다. 그 결과 Trigger pulse 는 Atmega8535 chip의 내장 타이머0를 이용, Overflow interrupt를 통해 일정 시간 간격마다 10uS 폭의 pulse 를 발생시키는 데 성공하였다. Echo pulse 폭의 측정은 단시간에 이해하기가 힘든 부분이 많아서 정확한 폭을 측정하는 것은 포기

하고 다음과 같은 방법을 고안해 개략적으로 폭의 길이에 비례하는 수치를 얻을 수 있었다.

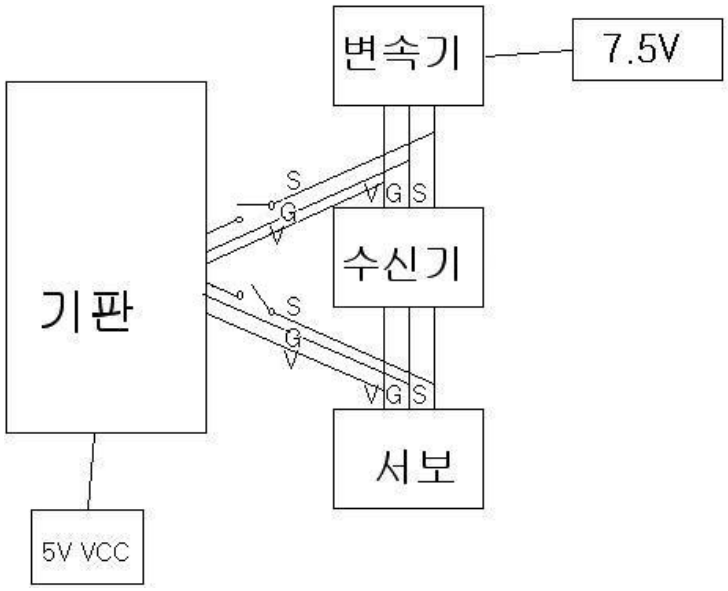
Echo pulse의 폭이 그 측정 거리에 비례해 100uS에서 25mS 까지 출력이 되므로 10uS 마다 Echo pulse 의 출력 값을 읽어 들여서 1일 경우 counting하는 방식을 취해봤더니 거리에 비례하여 증가하는 수치를 얻을 수 있었다. 상당히 rough 한 방식을 이용했음에도 이 센서 모듈이 제공하는 3cm 정도의 분해능을 얻을 수 있었다. 또한 작은 각의 변화에 대해서는 그 출력 값이 크게 영향을 받지 않았고 센서의 움직임에도 안정적으로 작동하였다. 30cm 이내의 근접거리에서도 잘 작동해 기존의 506nv에 비해 안정적인 거리 감지 능력을 보여주었다. 또 한가지 큰 장점은 동작 전원이 5v 인 것이었다. 기존의 센서가 12v에서 작동하여 전원 공급을 12v로 하고 정전압 regulator 를 통해서 회로에 5v를 인가하는 등 애로 사항이 있었던 반면에 srf04를 이용할 경우에는 5v 전원 하나 만으로도 regulator와 같은 추가적인 장치 없이 공통 전원을 공급할 수 있었다. 따라서 자연스럽게 1차 contest 준비 당시 전원의 공급문제로 발생했던 문제들은 더 이상 신경 쓸 필요성이 없어졌다.

센서를 통한 안정적 거리 측정 문제가 가장 큰 노력을 요구하긴 했지만 DC 모터의 제어 문제도 여전히 골칫거리였다. 다른 part들도 마찬가지였지만 독립적으로 제어를 했을 때는 별 문제가 되지 않던 것이 같이 연결할 때 정상적으로 작동하지 않는 경우가 많았던 것이다. DC 모터의 경우도 이와 같은 문제점이 있었다. 항상 동일한 속도로 동작하게끔 programming 했음에도 불구하고 그 동작 속도가 일정하지 못하였고 가끔 돌발적으로 rpm이 높아지는 현상을 발견하였다. 이를 해결하기 위해 rudder 의 움직임과 관계

없이 센서의 trigger pulse 를 발생시켰을 때 사용했던 내부 타이머 기능을 이용하여 고속 pwm(Pulse Wave Modulation) 파형을 발생시켜 다른 내부 알고리즘 동작과 독립적으로 항상 균일한 pulse파 입력을 통해 DC motor의 rpm을 안정적으로 유지시킬 수 있었다.

안정적인 자동/수동 전환을 위해서도 많은 노력을 기울였다.

11월 9일 센서를 차치한 자동제어를 위해 계속 새로운 연결을 시도해본 결과 다음과 같이 연결하였을 때 제어가 가장 안정적으로 되는 것을 발견하였다.



<그림 81> 최종 제어부 연결

처음의 연결과 다른 점은 기판과 변속기, 수신기, 서보모터에 연결되는 전원을 공통의 전원과 그라운드를 연결해 주는 것이다.

11월 10일 따로 센서테스트용 기판을 만들어서 센서가 제대로 작동한다는 것을 확인한 뒤, 기존의 기판에 센서를 연결하여 RS232를 통해 센서값을 읽

어 보았다. 그 결과 이상하게도 센서값이 제대로 읽히지 않았다. 그 원인은 아직도 파악하지 못하였다. 그래서 테스트용 기판에 약간의 커넥터와 연결을 추가하여 컨트롤러로 만들었다. 최종적으로 완성된 이 기판에서는 센서값을 잘 읽어 들이면서도 서보모터와 변속기의 제어가 완벽하게 이루어 졌다. 이제 적당한 센서 수치만 넣어주면 되는 단계에서 갑자기 기판이 작동하지 않았다. 제대로 작동 되던 기판이 아침 8시 30반에 작동이 되지 않아 당황스럽지 않을 수가 없었다. 어쩔 수 없이 프로그램과 연결을 처음부터 다시 해 보았지만 센서값이 읽히지 않았다. 한참을 씨름하다가 결국 기판에는 문제가 없고 RS232가 갑자기 고장 나서 센서값이 PC에 읽히지 않았을 뿐 센서를 포함한 자동제어는 제대로 작동하고 있었다. 컨테스트까지 시간이 얼마 남지 않아서 과전 앞 복도에서 가상의 테스트를 해본 후 정확한 수치를 입력하지 못하고 컨테스트에 나갔다.

5.3.2. 2차 Contest 결과 및 향후 과제

1차 Contest의 우수한 결과가 시운전을 통한 최종 조정 작업에 있었다면 2차 Contest의 실망스러운 결과는 자동제어를 통한 시운전의 수행을 해보지 못한 부분에 있었다고 판단된다.

알고리즘은 크게 문제가 없었다고 생각하지만 실제 조타각에 대한 선회 반경을 측정하지 못한 상태에서 '적당히' 수치를 대입한 탓에 실제 Contest에서 너무 과도한 반응을 보여 실패한 것이다. 2차 시도에서 조타각의 변화를 줄여서 다시 시도하였으나 그 조타각도 벽면과의 거리를 유지하면서 주행하기에는 너무도 큰 것이었다. 여러 차례 시운전을 통해서 피드백을 통해 최적 수치를 찾아야 함에도 시간의 제약으로 인해 이를 수행하지 못한 것은

가장 큰 패인으로 분석된다.

우리 조뿐만 아니라 모든 조들의 Contest 모습을 지켜보면서 제대로 된 무인 제어 주행을 위해서는 추가적인 장치의 보완 및 여러 이론 공부가 병행되어야겠구나 하고 생각했다.

먼저 제어 이론에 대한 공부이다. 여러 이론들이 있겠지만 PID 제어 이론에 대한 공부가 필요하다고 판단된다. 단순히 상한, 하한 값을 설정해놓고 양 극단에서만 러더의 방향을 수정하는 것으로는 안정적인 주행이 힘들다. 따라서 시스템의 90%이상을 제어하는데 쓰이는 방식인 PID 제어에 대해 공부를 좀 하여 러더의 각을 조정하는데 활용할 수 있을 것이다.

또한 거리 측정 센서만으로는 제어가 힘들 수밖에 없다. 선수의 방향과 진행 방향의 속력 및 회전 속도 등이 안정적 제어를 위해서는 필수 Input이라고 할 수 있다. 변속기 제어를 통해 속력은 일정하게 유지한다고 해도 선수의 방향과 회전 속도 등은 거리 측정 센서만을 가지고는 알기 힘들다. 측면에 센서 2개를 장치한다고 하더라도 초음파 센서의 특성상 그 정확한 방향을 구하기란 힘들다. 따라서 추가적으로 방향 제어에 이용할 수 있는 자이로 센서 등을 이용해 볼 수 있을 것이다. 물론 이의 사용은 초음파 센서의 제어보다는 훨씬 힘든 작업이 될 것이다. 하지만 정말 노력과 의지만 있다면 도전 해볼만한 과제라 생각한다.

6. 역할분담 및 세부일정

<표 18> 역할분담

선체부	선형설계	김두용, 김영현
	구획배치, 도면생성	김두용, 김영현
	Nesting&cutting	이주현, 심훈섭, 노재욱
	선체조립	이주현, 심훈섭, 노재욱
	선체마감작업	모두
제어부	회로설계	이주현, 심훈섭, 노재욱
	회로제작	이주현, 심훈섭, 노재욱
	무인주행프로그래밍	김두용, 김영현
시험주행&조정		모두

<표 19> 세부일정

		10.11	10.28	11.11
선체부	선형설계			
	구획배치, 도면생성			
	Nesting&cutting			
	선체조립			
	선체마감작업			
제어부	회로설계			
	회로제작			
	무인주행프로그래밍			
시험주행&조정				

7. 재료비

품목	수량(개)	단가(원)	금액(원)
니퍼	1	10000	10000
ml701	1	10000	10000
인두대	1	10000	10000
납	1	2000	2000
led	10	30	300
s/20	10	200	2000
기판 25*34	2	2400	4800
hc244	3	300	900
저항	1	100	100
콘덴서	13	50	650
크리스탈	3	300	900
콘넥터	2	65	130
4148	1	100	100
db25cbc	1	1500	1500
저항100k	100	10	1000
atmega8535	5	2600	13000
소켓			1120
usb a tape	1	150	150
headpin 1x40	1	60	60

yk 16 v 10m	2	20	40
다이오드4148	10	10	100
d-sub기판	1	1600	1600
인두기	1	4500	4500
기판 **35*55**	1	4200	4200
bc4aaal	2	800	1600
lro3cp	10	250	2500
5264-2p	6	20	120
5268-2p	6	20	120
가변저항	3	200	600
5264 wire	12	30	360
우드락 **5t**			15500
mr-422	1	24200	24200
45도 칼	1	1500	1500
30도 칼	1	7500	7500
자동우드락커터기	1	22000	22000
우드락본드	3	1200	3600
에폭시본드	2	5500	11000
붓	3	2400	7200
540s 모터	1	10200	10200
스크류샤프트	1	4300	4300
기어박스	1	4300	4300

기어	1	1700	1700
드라이브샤프트	1	4300	4300
기어 어댑터	1	2600	2600
스크류 샤프트	1	3400	3400
attack 2er(am)	1	66300	66300
propeller (d20*p1.2)	1	9600	9600
7.2v 배터리	1	20000	20000
모터콘넥터(수)	1	2400	2400
840f 볼베어링	1	1600	1600
변속기	1	65700	65700
퍼티	1	25000	25000
테이프	1	1500	1500
사포	5	300	1500
글루건 심	10	100	1000
isp cable	1	11000	11000
초음파센서 **506nv**	3	75000	225000
페인트 외	1	68500	68500
에폭시본드	2	5500	11000
발사나무	1	4900	4900
rs232 보드 **22v**용	1	8800	8800
초음파센서모듈#1	1	43500	43500
건전지			35550

총합			800600

8. 후기

♦ 김두용

드디어 1차, 2차 contest를 끝내고 최종보고서 제출만 앞두고 있다. 작업들 어느 하나도 쉽지 않았고 물론 방법을 미리 알고 있지도 못했다. 매일 밤을 지새운 것은 아니지만 주요 작업이 있을 때마다 조원 들과 새벽 늦게까지 혹은 밤을 꼬박 새며 작업을 할 때는 조선해양공학계획 수업의 '악명'을 몸소 체험할 수 있었다. 특히 contest 직전 sensor를 통한 자동제어를 위한 작업을 할 때면 거의 잠을 안 자가면서 작업을 해도 실패하기도 하고 작업에 진전을 이루지 못하기도 했다. 또 거의 다 된 듯하다가 갑자기 전자제품이 망가지면서 작업을 원점으로 돌려놓기도 했었다. 그 때마다 어설픈 조장의 지시와 반복된 작업에 불평 없이 잘 따라주고 도와줬던 조원 들에게 먼저 고마움 마음을 전하고 싶다.

첫 수업시간에 이번 학기의 과제를 들으면서 막연한 마음만 들었는데, 차근차근 조교님이 주신 소 과제들을 해결하며 그 틀이 잡혀가는 것을 느끼면서 우리도 할 수 있구나 하는 자신감도 얻을 수 있었다. 첫 Proposal 때 과

제에 대한 이해도가 모자라서 처음부터 선형을 바꾸는 시행 착오도 있었지만 그 이후 대체로 무난하게 Project가 진행되어 다행이었다.

이 수업이 단순히 모형선박을 제작하는 skill을 익히는데 그치는 것이 아니라 공학인으로서 갖추어야 할 팀원간의 의사소통 능력과 팀 단위 협업능력 그리고 조장으로서 Presentation skill과 조원들을 이끌어나가는 leadership 등을 배양할 수 있는 계기가 되었던 것 같다. 하지만 그보다 더 좋았던 점은 그 전에는 알지 못했던 동기들끼리 서로 인사를 나누고 친해지게 되는 계기가 되었다는 점이다. 아마도 수업 처음에 조를 나눌 때 마음에 맞는 사람끼리 조를 만들었다면 지금과 같이 동기들이 친해질 수 없었을 것이다. 비록 어떤 조는 갈등과 불화로 인해 싸우는 사람들도 있었지만 결과적으로 볼 때 그 동안 얼굴만 알았지 잘 모르고 지냈던 사람들끼리 친해지는 계기를 바로 이 수업이 만들었다고 생각한다. 또 동기들간의 동질감을 느끼게 해주고 조선과라는 곳에 소속감을 심어주었다고 생각한다.

작업하는데 있어 물론 선체부분도 어려웠지만 제어부가 매우 힘들었다. 선체는 작년 선배들이 한 것을 유추해가며 작업을 진행해 나갈 수 있었지만 제어부는 배경지식도 없어 처음 시작부터 애를 먹었다. Avr에 대한 공부는 물론이고 초음파 sensor에 대해서도 많은 공부를 했다. 1차 contest 전에는 program상으로 자동 수동을 변환하려고 했지만 어려움은 물론이고 그 부분에 시간을 들이는 것이 경제적으로 비효율성을 느껴 손쉽게 기계적으로 자동수동 변환하는 방법을 택했다. 2차 contest 때는 자동제어를 보여야 했기 때문에 1주일 내내 sensor만 붙잡고 매달려 있었다. 처음에 샀던 초음파 sensor를 가지고 계속 연구했지만 결과적으로 우리 project에 불가능하다는 판단을 하고 contest 전날인 11월 10일 금요일에 다른 초음파 sensor로 바

꿨다. 사실 지금 생각해보면 정말 무모한 도전이었다. 기존에 사용하던 sensor보다 사용법도 배로 복잡하였고 sensor사용법을 익히고 그에 맞게 작동하게 하는데 시간이 너무 부족했기 때문이다. 11월 11일 Contest 당일 새벽 1시에 sensor를 작동시키는데 성공하고 신호를 주고 받을 수 있게 되었을 때의 기쁨과 환희란 이루 다 말 할 수가 없다. 그 때 함께 곁에서 자리를 지켜주면 나를 도와주었던 조원들이 너무나도 감사하다. 그들의 도움이 없었다면 그 때 그 작업을 성공시키지 못했을 것이다. 하지만 예상외의 문제가 우리의 발목을 잡았다. RS232가 고장이 났는데 그것을 알지 못하고 우리의 programming 의 실수로 생각해서 작업을 처음부터 다시 하느라 많은 시간을 낭비하고 말았다. 그 때 RS232가 고장 난 것을 알아 채지 못했다면 실컷 성공하고 contest를 망칠뻔했다. 좀전의 문제로 contest 직전에 자동제어 system을 완성해서 수조에서 시운전을 못해 contest 때 애를 많이 먹었다. 하지만 결과적으로는 좋은 결과가 나와 우리 조원들의 수고가 보상받는 것 같아 매우 기분이 좋았다. 물론 다른 조들도 열심히 했겠지만 우리조원들이 누구보다도 열심히 한 것을 알기에 너무나도 감사하고 이들과 함께 조가 된 것에 대해 다행이라고 생각한다. 이번 수업을 통해 조선공학도로서 좋은 경험을 한 것도 좋았지만 동기들간에 힘들게 함께 작업하며 고난을 이기고 함께 기쁨을 나눌 수 있었던 점이 좋았다. 마지막으로 작업에 많은 도움을 주신 이재승 조교님과 이규열 교수님께 감사의 말씀을 드리고 싶다. 이번 학기는 나에게 절대 잊을 수 없는 소중한 추억으로 남을 것이다.

♦ 김영현

내가 서울대에 교환학생으로 와서 ‘조선해양공학계획’이란 과목에 대해 내가 아는 건 그냥 단지 배를 만들어서 컨테스트를 한다는 것 뿐 이였다.

컨테스트라 하면 문뜩 1학년 때 울산대에서 호버크래프트를 만들어 컨테스트를 해봤던 일이 떠올랐다. 그래서 이런 종류의 수업을 한번 해봤다는 경험에서였는지 처음엔 쉬울 것 같기도 하고 아무튼 도무지 감이 잡히질 않았고 잠깐이지만 자만도 했었다.

하지만 첫 수업을 들으면서 ‘앗!! 힘들겠다.’ 라는 생각이 뇌리를 스쳤다.

첫 수업 내용에는 배를 직접 설계하고 도면을 출력해 제작하여 자동제어로 움직이는 모형 선박으로 컨테스트를 한다는 것 이였다.

그리고 첫 수업 때 교수님이 보여주신 기사가 생각 나서 집에 와서 인터넷으로 관련 기사를 찾아보았다. 그 기사에는 서울대 조선해양공학과 2학년 2학기 조선해양공학계획 수업을 얘기하면서 ‘보약 먹고 수업 들어라’, ‘이

조선해양공학계획 수업은 체감 학점이 10학점이고 한 학기 동안 이 한 과목만 듣는 것 같다'는 글이 보였다. 점점 이게 아니다라는 생각과 동시에 걱정이 앞서기 시작 했고 그 후에 계속 수업을 들으면서 앞으로의 일들이 나에게 더 큰 두려움과 걱정으로 가득 찼다. 간단히 스위치를 켜서 속도경쟁을 했던 1학년때 보다 훨씬 더 복잡하고 어려웠다. Ezship프로그램을 이용해 설계 도면을 출력해서 네스팅, 커팅작업 후 제작, 그리고 자동제어까지 말만 들어도 거부감이 밀려왔다. 정말 첫 시간에 보여준 기사 말대로 이번 학기는 조선해양공학계획 하나만 듣는 것 같이 느껴졌다. 첫 수업 후 무작위로 같은 조가 되었지만 서로 인사 정도도 나눌 틈 없이 Proposal 1차 발표를 준비하게 되었다.

준비 과정은 생각보다 어려웠고 나름대로 정성껏 준비한 보고서였지만 교수님은 우리에게 '전면 재수정' 이란 말을 하시고 그렇게 06년 조선공학계획 1차 Proposal 발표를 마쳤다.

그래서 우리 조는 보고서를 한번 더 쓰게 되었다. 초반에 쓴맛을 봐서 그런지 그 다음 선형설계서부터 제작에 이르기 까지는 신중하고 최대한 신경을 써서 준비를 하여 중간발표 그리고 마지막 Proposal까지 작성하게 되었다.

하지만 1차 발표 이후에 선형 설계를 하기위해 필요한 Ezship프로그램에 적응하기 까지 시간이 좀 많이 걸렸고, 도면 출력 후 모형 배 조립이 시작하면 시간이 많이 안 걸릴 거라 생각했는데 생각보다 선체조립과 도장 쪽 에서도 상당한 시간의 투자와 힘든 작업의 연속 이였다. 그리고 특히 제어부 쪽에서는 정말 전기쪽 지식이 거의 없는 쪽에서 시작했는데 내가 제어부 파트를 맡은 건 아니 였지만 이번 수업과 여러 가변 저항, led 등을 제작 해 보면서 자동제어구동에 필요한 전기제어파트에 대해서 어느 정도 알 수 있는 계기

가 되었다.

하지만 컨테스트 전날 까지 고생을 했던 부분은 예상치 못했던 센서 부분에서였다. 프로그랭밍 등 센서구동에 필요한 모든 부분이 완벽 하다고 판단 했지만 무슨 이유에서인지 센서가 제대로 작동을 하지 않는 것이였다.

결국 나중에서야 센서가 불량이라는 것도 알았고 무엇보다 그 초음파 센서는 우리에게 적합하지 않은 센서라고 판단을 했다.

어쨌든 그렇게 단 한 조도 자동제어를 완성하지 못한 채 센서 부분을 제외하고서 1차 컨테스트에 나가게 되었다. 지금까지 배를 만드는 과정에서 많은 것을 배운 것에 만족했고 제발 물에 가라앉지만 않았으면 하는 마음 뿐 이였다 하지만 1차 컨테스트 당일 날 이상하게 우리 배가 1등 했으면 하는 생각을 하면서 수조로 들어섰다. 그 때 조장이신 두용이 형이 나에게 컨트롤을 맡기셨다. 난 당황했고 잘할 수 있을지 걱정도 되었다. 하지만 내가 시운전도 해봤으니 당연한 결과로 받아들이고 컨트롤러를 잡았다.

그런데 1차 컨테스트에서 우리배가 직진성과 조종성에서 다른조의 배 보다 우수한 성적을 거두어 1등을 했다. 믿을 수 없었고 다른 조도 열심히 했지만 우리 조가 더 열심히 한 결과라 생각했다. 하지만 자동제어에서는 아무도 완성하지 못하였기에 2차 컨테스트까지 하게 되었다. 처음에 샀던 센서의 불량으로 우리 조는 다른 센서를 이용해서 하기로 하고 조원들 모두가 새로운 센서에 올인을 하고 밤을 세어가며 컨테스트 하기 30분전까지 구동이 되는 것을 육안으로 확인했다. 하지만 역시 시운전을 하지 못한 것이 문제였다. 우리에게 조금의 시간이 더 있었으면 문제점을 확인하고 충분히 성공할 수 있었을 것이라 아직도 생각한다. 비록 2차 컨테스트에서는 성공하지 못했지만 우리 조가 2등을 1점 차이로 제치고 1등을 하는 영광을 가지게 되었다.

말할 수 없이 너무나 기뻤고 마지막까지 같이 작업 한 조원들이 고맙고 자랑스러웠다.

그리고 처음에 서울대 교환학생으로 왔기 때문에 06학번 이지만 원래 03학번인 나에게는 05학번들과 한 조를 이룬다는 게 처음엔 약간 부담 이였다. 나보다 다 나이가 어리고 내가 교환 학생이라 친해지기 힘들 거라는 생각이 앞서서 그랬던 것 같다. 하지만 우리 조의 조장이자 팀의 기둥인 두용이 형이 계셨고 지금은 조원인 재욱, 훈섭, 주현이와 많이 친해져서 그런지 그런 걱정은 온데 간데 없이 사라졌다. 그리고 지금은 우리 조원 뿐만 아니라 이 과목을 같이 들은 친구, 형, 동생들인 동시에 경쟁자였던 많은 사람들을 알게 되었고 정말 많이 친해 지게 되었다. 정말 이 과목을 신청한 것을 잘했다는 생각이 들었고 정말 이번 추억이 대학생으로써 2학년 2학기가 아닌 내 인생에서 23살이 나에게 정말 특별하고 무엇과도 바꿀 수 없는 추억이 될 거 같다.

마지막으로 이규열 교수님과 이 수업과 관련되어 모형 선박 제작에 신경 써 주신 모든 조교님들과 선배님께 감사 드립니다.

♦ 노재욱

서울대 공포의 수업 '조선해양공학계획'. 인터넷 신문의 한 페이지를 장식한 이 수업에 대한 얘기는 선배들과의 대화에서 항상 화제거리였고 지금은 우리들의 입에 오르락내리락한다. 이 수업에 대해 처음 들은 것은 선배들을 통해서가 아닌 1학년 때 수강한 이규열 교수님 수업 '신입생 세미나'를 통해서였다. 그 때는 우리 2년 선배들의 수업결과 보며 수업에 대해 알게 되었다. 사실 그 때만 해도 수업에 강한 흥미를 느끼고 흥분하여 항공모함을 만들어 자하연에 띄어 보겠다는 큰 꿈을 가지고 있었다. 비록 항공모함은 아니지만 다른 어느 조보다 선형이 매끈하고 예쁜 VLCC 한 척을 우리 손으로 만들어 냈다. 학기 초의 그 막막함과 답답함을 느꼈던 것이 엊그제 같은데 벌써 contest까지 끝내고 시상식까지 치렀다. 정말 어떻게 시간이 지났는지 모를 정도로 빨리 지나간 것 같다.

'선박제어자동화', 우리의 이번 주제가 KAIST 대학원 Lab에 의뢰되어있는

것을 두용이형이 보신적이 있다고 했다. 사실 수면에서의 자동제어가 말이 쉽지 그 분야의 전공도 아닌 조선과의 학부 2년생이 하기에는 매우 어려운 과제였다. 하지만 지금 우리는 그 과제를 다른 조들에 비해 가장 성공적으로 해냈다. 어떻게 해냈는지 지금 생각해 보면 그저 놀라울 따름이다. 센서 작동과 코딩에 대한 문제를 실질적으로 해결하신 조장 형이 너무나도 자랑스럽고 멋지게 생각한다.

우리의 작업들을 다시 생각해보면 별거 아닌 것 같지만 만드는 과정 당시 우리에겐 엄청난 시련과 고난이었다. 다른 조 보다 작업을 한 템포씩 빨리 진행했던 우리 조는 과정과정마다 난관에 부딪쳐 실수하고 고민하며 방법을 하나씩 터득해 나갔다. 배 만드는 작업은 마치 하나의 산을 넘으면 그 뒤에 또 다른 큰 산이 나오듯이 일이 점점 어려워지는 작업이었다. 처음 부재를 자르고 골격만 다 끼워 맞추고 마치 배 작업이 다 끝난 것이라 생각했던 것을 지금 생각해보니 피식 웃음이 나온다. 골격을 맞추고 곡선이 있는 부분을 얇은 우드락으로 외판을 만드는 작업이란 정말 쉽지 않은 작업이었다. 우리 조 의 선형은 다른 조와 달리 선미와 선수가 곡선으로 되어있어 외판작업을 하는데 조금 어려움을 겪었다. 더군다나 외판을 만들 때 테이프를 써도 되는지 몰라서 일일이 풀로 힘들게 붙였던 것이 생각난다. 외판 작업을 맞추고 이번 프로젝트에서 주안점이 선형이 아니라는 점에 연신 감사했다. 그리고 작업실 작은방에 2년전 선배들이 만들었던 삼동선의 살벌하게 매끈한 선형을 볼 때마다 감탄을 하지 않을 수 없었다. 외판을 두르고 처음 퍼티칠을 할 때 방법을 몰라 대충 발랐던 것이 울퉁불퉁하게 굳어져서 나중에 힘들게 사포질을 해야 했다. 그 다음에 방수처리와 외판강화를 위해 에폭시를 먼저 발라주고 페인트를 발랐어야 했는데 페인트를 먼저 바르고 에폭시를 발라 에

폭시가 녹기도 했다. 선체 제작 후 추진부를 제작할 때 제작법이 있는 것도 아니라서 선배들이 만들어 놓은 배를 보면서 하나하나 유추해가며 만들었다. 이런 저런 방법을 생각하면서 겨우 만들었다. 그리곤 우리 조가 만든 추진부가 다른 조들의 제작방법으로 이어져갔다. 직접 손으로 나무를 깎아 Rudder를 만들고 배에 달아 조종기로 처음 조종했을 때 너무 신기하고 감동적이었다.

그리고 발표 전 시운전 할 때의 그 감동은 지금도 기억 난다. 우리가 만든 배가 내 손에 의해 움직이는 것을 보니 신기하기도 하고 다행스럽기도 하고 기분이 참 묘했다. 하지만 수동제어만 되고 실제적인 주제 '자동제어'는 되지 않았다. 모든 조가 자동제어에 실패하여 할 수 없이 1차 contest 때는 수동제어 성능만 판단하고 2차 contest 때 sensor를 이용한 자동제어를 평가하기로 했다. 우리 조는 contest 전날 배가 직진으로 가게 하는 rudder 각을 찾아 그 때의 각을 rudder 중앙으로 교정하는 작업을 했었다. 그 덕분에 우리 조는 1차 contest 때 1등을 하게 되었다. 그리고 2차 때 우여곡절끝에 contest 당일 날 새벽에 센서 작동을 성공시키면서 최종으로 1등으로 마쳤다. 우리 조가 1등을 한 것도 물론 기뻤지만 그 것보다도 자동제어라는 어려운 주제를 우리가 해냈다는 점에 너무 기뻤다. 그 동안 함께 밤새며 작업했던 우리 조원들에게 너무 감사하고 작업하는데 많이 도움을 주신 조교님들과 교수님께 감사하다. 내 인생에 단 한번 있을 소중한 추억과 경험 절대 잊지 못 할 것이다.

♦ 심훈섭

보약 먹고 듣는 수업인 조선해양공학계획. 모두들 두려움에 떠는 과목이다. 하지만 나는 내심 조선해양공학계획 수업이 기다려졌다. 모형 만들기라면 아무리 힘들어도 즐겁게 할 수 있을 것 같았기 때문이다. 왜냐하면 나는 고등학교 1학년 때부터 취미활동으로 모형선박 특히 군함을 만들어 왔기 때문이다. 사실 조선해양공학과에 진학한 계기도 모형으로 만들던 군함을 직접 만들어 보고자 하는 마음에서 였다. 이런 이유로 첫날 수업 하는 시간까지 들뜬 마음으로 기대하고 있었다.

하지만 막상 강의 계획서를 들여다 보니 만만치 않은 부분이 한 둘이 아니었다. 대충 그냥 만드는 모형선박이 아니라 조선공학에 사용되는 많은 계수들을 직접 계산하고 수치화 시킨 후에 설계 프로그램을 사용하는 진짜 배를 만드는 것 이었다. 게다가 “배” 자체 외에도 부가 기능으로 자체 주행기능, 장애물 회피기능, 유사시 리모트 컨트롤러로 전환하는 기능까지 이전에

는 생각하지도 못했던 조선공학 이외의 전기공학과 컴퓨터 프로그래밍 쪽의 지식을 상당한 양 필요로 한다.

본격적으로 계획수업이 진행됨에 따라 “보약 먹어라” 라는 말이 현실로 다가왔다. 제어부 자제를 구하기 위해 직접 용산으로 구로로 발품을 팔면서 직접 현장을 체험했다. 자제를 구하고 나서는 난생 처음으로 인두를 들고 납땜을 했다. 냄새도 빠지지 않는 지하실에서 코끝을 찡하게 찌르는 납 냄새는 종종 두통을 유발하기도 했다. 납땜 작업이 끝나자 선체제작에 들어갔다. 선체의 프레임을 만드는 과정은 “노가다”의 연속이었다. 같은 모양을 자르고, 자르고, 또 자르고.. 커팅이 완료되었을 때 는 마치 프로젝트가 끝난 듯한 느낌마저 들었다.

우리 조는 시험도 뒤로하고 커팅이 완료되자 마자 퍼티와 에폭시 페인트 작업에 들어갔다. 다른 조보다 2~3일 빠른 속도로 진행했기 때문에 여러 에로사항을 직접 몸으로 겪으면서 배를 만들어갔다. 비록 만드는 과정은 어려웠지만 먼저 접한 문제를 스스로 해결하는 도전정신이 모르는 사이에 몸에 베고있었다.

추진부 설치가 끝나고 방수작업도 마무리 지었다. 수조동에서 시운전을 했다. 잔 고장 없이 물살을 가르는 배를 보며 지난 2달간의 시간이 머릿속을 스쳐지나갔다.

2학기 수업시작 후 2달, 주로 따지면 10주간의 시간이 훌쩍 흘러 어느덧 모형선박이 완성단계에 다다랐다. 잠시 눈을 감고 생각해보니 2달 동안의 일들이 찰나처럼 스쳐간다.

아무것도 없던 곳에서 뼈대가 세워지고 외판이 덮힌다. 얼마 되지 않아 퍼티가 칠해지고. 독한 냄새를 풍기는 빨간 페인트가 배의 외판을 덮는다.

드릴로 샤프트 구멍과 러더 구멍을 뚫으니 제법 배다운 모양이 잡힌다. 기어를 조립하고 모터를 장착한다. 방수처리 마무리를 하고 시험주행마저 마쳤다.

하지만 1차 콘테스트를 앞두고 시험주행을 한 결과 우리가 선택한 센서에 중대한 문제가 있음이 발견되었다. 지상에서는 정상적으로 작동하던 센서가 물에 띄우자 “바보”가 되어 버렸다. 센서값을 제대로 읽지 못하자 배는 벽면을 따라가지 못하고 갈팡질팡 지그제그로 가는 것이었다.

긴급 대책회의가 벌어졌다. 다른 조들 모두 같은 현상으로 낙담하고 있는 상황이었다. 그런 상황에서 우리 조는 일단 이번 콘테스트는 자동제어가 불가능 하다고 판단 수동제어와 러더의 중심을 정확히 잡는 것으로 방향을 잡았고 밤 늦게까지 수동제어의 조종성을 확보하기 위해 시운전을 하였고 거의 완벽한 직진성과 선회성을 확보 할 수 있었다.

그러나 시험주행 이후 선체운반 도중 선체외벽에 금이 가는 사고가 발생하여 긴급히 퍼티와 에폭시 페인트칠을 다시 하는 작업을 하였다. 그 결과 방수처리도 더 잘되고 선형도 미끈하게 빠지게 되었다.

대망의 1차 콘테스트가 있던 11월 11일, 많은 시행착오와 선체에 금이 가는 사고도 발생했지만 조장형이 콘테스트의 방향을 정확히 예측하여 1차 콘테스트에서 7조들 중 가장 좋은 직진성과 선회성을 확보 할 수 있었고 1차 콘테스트 1위를 할 수 있었다.

1차 콘테스트를 1위로 끝냈지만 아직 자동제어가 완성되지 못했기 때문에 안심하고 있을 수는 없었다. 핵심과제인 자동제어 부분은 아무리 수정을 가해도 제대로 작동되지 않았다. 자동제어의 핵심인 센서 자체가 진동에 취약했기 때문에 우리의 기술로는 그 부분을 보정하기 어려웠다. 몇 일간의 시행

착오를 거치고도 더 이상 기존 센서로는 불가능하다고 판단 추가비용의 부담이 갔지만 프로젝트 수행을 위해 다른 센서를 찾아나섰다. 콘테스트 하루 전에 새로운 센서를 문래에 가서 구입한 후에 센서에 맞게 새로운 기판과 전압장치를 제작하였다.

11월 18일 새벽 2시경에 드디어 우리가 새로 구입한 센서가 정확한 값을 출력해 내었다. 거리에 따라서 완벽한 값을 출력함은 물론이고 진동에도 전혀 값의 영향을 받지 않았다. 이제 우리의 알고리즘에 맞게만 프로그램을 한다면 2달동안 매달렸던 “자동제어와 수동제어가 가능한 모형선박”을 완성시킬 수 있었다.

조장형이 프로그램을 짜는 동안 지하 작업실과 2층 과전산실을 들락날락거리며 프로그램에 맞는 기판을 만들고 수정하고 추가하고... 새벽시간대에 잠도 못자서 피곤할 만도 하지만 거의 다 완성되었다는 기대감에 잠오는 줄도 모르고 바쁘게 뛰었다.

드디어 11월 18일 아침 8시 30분 완벽한 성능이 발휘되는 프로그램이 완성 되었다. 복도에서 실험한 결과 전방센서는 약 2미터 거리에 벽이 감지되었을때 왼쪽으로 30도 가량 러더를 틀었고 오른쪽 센서는 벽이 다가오면 왼쪽으로, 멀어지면 오른쪽으로 러더를 돌렸다. 다른 조들도 우리가 복도 실험에서 성공하는 모습을 보고 프로젝트의 성공을 예감한 듯 환호성을 질렀다. 단지 시간이 부족해서 수조에서 직접 실험해 보지 못한 것이 마음에 걸렸지만 성공을 눈으로 확인하자 이미 콘테스트는 끝났다는 생각마저도 하게 되었다.

콘테스트가 시작되고 1조부터 배를 띄우기 시작하였다. 우리 조는 7조로 마지막 순서였는데 우리 앞 조들이 모두 자동제어에 실패했다. 6조 모두 실

패하자 당연 관심을 우리 배에 쏠렸다. 복도에서의 성공도 그렇고 우리 조원들도 내심 우리 배는 완벽하게 자동제어를 해낼 것으로 믿고 있었다.

하지만 막상 운전을 시작하자 배가 왼쪽으로 한번 틀고 오른쪽으로 한번 틀고 부터는 계속 원을 그리며 빙빙 도는 것이었다. 순간 머리가 띵해졌지만 문제 분석에 들어갔다. 센서의 값은 러더가 오른쪽으로 튼 것을 보아 정확히 읽혔지만 문제는 러더의 트는 각도와 틀여져 있는 시간, 그리고 프로펠러의 회전 속도였다. 센서가 값을 읽고 판단하는 시간보다 배의 관성이 더 커서 제대로 제어가 되지 않았던 것이었다.

시운전 해볼 시간이 반나절만 더 있었다면 이정도 문제는 프로그램의 조정을 통해 극복할 수 있었지만 그러지 못한 것이 너무나 아쉽게 느껴졌다. 조교님도 우리 배를 보고 이거는 성공한 것이나 마찬가지 라고 위로해 주셨지만 완성 직전의 단계에서 그만 무릎을 꿇었다는게 너무나 아쉽고 반나절의 시간이 너무나 안타까웠다.

최종 합산결과 우리 레노바티아가 1등을 차지했다. 1등이란 숫자도 물론 중요하다. 하지만 그것보다 더 중요한 것은 70여일의 시간동안 우리 5명이 같이 밤새가면서 배를 만들고 고민을 하고 노력했다는 것이 더 중요한 것 같다. 2년간의 대학 생활을 하면서 이번 계획 같은 수업은 없었던 것 같다. 대부분의 수업이 대충 수업 듣고 시험보고 대출도 하고 그런 것이었지만 계획 수업은 참여와 팀플이라는 앞으로의 사회생활에 필요한 것들을 피부로 느끼게 해준 수업이었다.

보약 먹고 들어야 하는 수업을 보약을 먹지도 않고 무사히 모형선박을 제작했다는 거에 대해 감사하며 2달 동안 꾸준히 작업에 참여한 7조 ; 김두용, 김영현 이주현 노재욱 모두 정말 잊지 못할 추억을 함께해 감사하다. 특히

우리 조를 확실 한 리더쉽으로 이끌어 주신 조장 김두용 형님께 수고하셨다는 말을 전하고 싶다.

♦ 이주현

조선해양공학계획의 모형선박제작 프로젝트의 최종 콘테스트가 지난 11월 18일을 끝으로 대장정의 끝을 맺었다. 그 동안 휴일이 없을 뿐만 아니라 중간고사기간, 다른 과목 수업시간, 심지어 날짜 개념도 없이 선체 제작과 회로 제작을 하였다.

매번 배를 띄우면서 느끼는 거지만 물살을 해치고 나아가는 배를 보면 지난 2달간의 노력이 보상될 만큼의 감동과 뿌듯함이 느껴지곤 한다. 정말 지난2달 동안 해왔던 일들은 그 동안 살면서 해보지 못했던 다양한 경험들이었던 것 같다.

처음 프로젝트가 정해지고 조 발표가 나왔을 때 매우 다행이라는 생각을

했다. 우리 조의 조장이신 두용이 형은 작년부터 우리과의 소위 “에이스”였기 때문에 믿음이 가고 이번 프로젝트가 잘 될 것 같다는 느낌이 들었었다. 다른 조들 보다도 서로 아는 사람들도 많고 해서 다른 조들이 서로 어색해 하는 사이에 우리 조는 작업에 집중 할 수 있었다.

이런 좋은 상황에서 인지 우리 조는 다른 조들 보다 빨리 작업을 진행 할 수 있었다. 초기 선형이 이지스 함에서 VLCC로 바뀌는 우여곡절도 겪었지만 그런 것들을 커버할 정도의 작업량과 작업속도를 우리 는 가지고 있었다. 다른 조가 겨우 이지쉽 프로그램을 이용해 선체 도면을 뽑기 시작 했을 때 이미 네스팅과 컷팅을 하고 있었다. 빠른 속도로 선체가 제작되어 가고 있을 즈음에 제어부 컨트롤러 제작을 위한 기초 교육으로 간단한 회로에 대한 과제가 나왔다.

이 때부터 본격적으로 제어부에 참여하게 된 나는 정말 아무것도 모르는 상황에서 롬라이터와 LED 회로, 서보모터 제어회로를 제작해야 했다. 조교님이 올려주신 회로도를 보면서 하나씩 더듬더듬 납땜을 하고 전선을 연결해 나갔다. 난생 처음 하는 작업이라 잘될 턱이 없었다. 분명 회로도와 똑같이 만든다고 만들었는데 롬 라이터는 작동을 안하고 자꾸 에러가 뜨는 것이었다. 회로에 대한 기초 지식이 부족해서 어디가 문제인지 파악할 수도 없었다. 결국 처음부터 다시 제작을 하기로 하고 납땜부터 천천히 하나하나씩 해나갔다. 신기하게도 분명 그전것과 같은 방법으로 만들었는데 이번에는 제대로 작동을 하는 것이었다. 왜 그러는지 이유를 파악할 새도 없이 LED와 서보 회로를 제작했다. 직접 부딪혀 보면서 배워가니 이론으로 공부하는 것 만큼의 지식이 쌓이는 것을 느낄 수 있었고 과제를 담당하면서 실제 배의 컨트롤러 제작도 주도적인 부분을 담당해서 하게 되었다.

실제 모형배를 자동으로 제어할 컨트롤러 자체를 만드는 것을 앞에서 겪은 많은 시행착오 덕분에 쉽게 만들 수는 있었다.하지만 센서가 작동을 하지 않아 센서의 문제인지 기판의 문제인지 점점 혼란이 오기 시작했다. 결국 컨트롤러 기판만 5개를 제작하는 우여곡절을 겪게 되었다.

첫 번째 컨트롤러는 1차 테스트가 있던 11월 11일의 일주일 전일 11월 3일날 완성되었다. 이 기판을 이용하여 테스트 해본 결과 센서 값이 거리에 따른 특정값이 아니라 랜덤으로 출력되었다. 문제가 발생한 것이다 예상대로였다면 센서 값이 거리에 따라 다르게 출력되어야 하는데 비례가 맞지 않았다. 고민과 연구..그리고 밤샘이 계속 되었다. 콘테스트 날은 다가오는데 결과가 좀처럼 나오지 않았다. 그러던 중 6조 기영이 형이 센서의 전압과 기판의 전압이 맞지 않는 다는 사실을 알아냈다. 이에 전압을 낮춰주는 regulator를 기판에 붙여넣은 2번째 기판을 제작하였다.

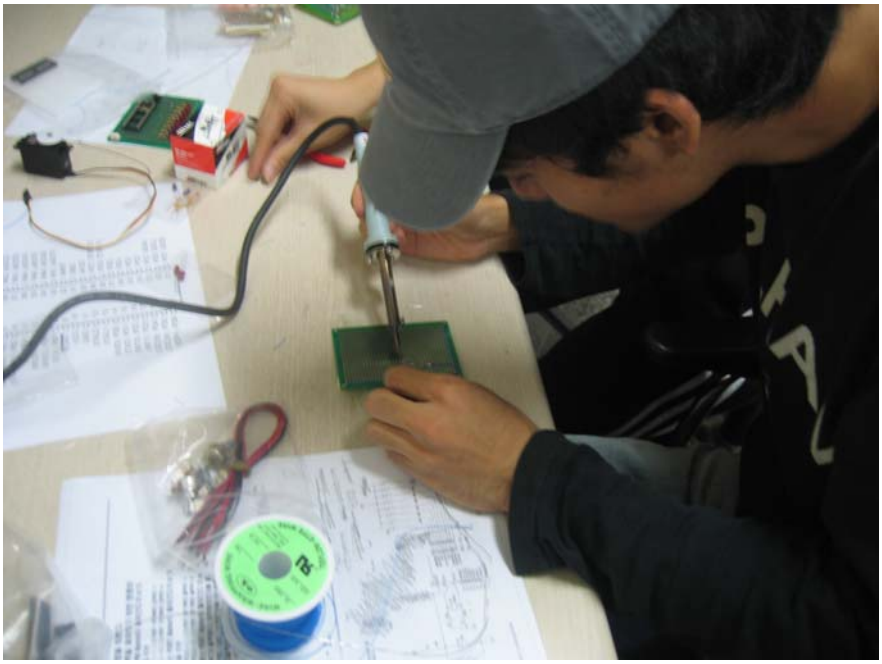
2번째 기판 까지는 자동/수동 전환을 소프트웨어를 이용해서 하려고 했다. 그러나 실제로 제작을 해본 결과 서보모터와 변속기를 제어하는 기판과, 송신기가 켜짐을 인식하는 기판을 각각 따로 제작할 수 는 있었지만 그 두가지 기능을 합한 하나의 기판을 만들자 제대로 작동하지 않았다. 이 것은 신호가 다른 선으로 역류했을 것이라고 판단하고 다이오드를 다는 등의 수정을 해 보았으나 다이오드를 달자 회로전체의 전압과 전류가 달라지는 것 같아 전기적인 지식이 부족해 소프트웨어를 통한 자동/수동 제어는 남은 시간상 힘들 것 같아 포기 하게 되었다.

다음으로 수동과 자동제어의 전환을 스위치를 통해 하드웨어적인 방법으로 하는 세 번째 기판 제작에 들어갔다. 이 때에는 서보모터와 변속기에서 신호선만을 기판에 연결하였다. 그러나 이유를 알 수 없이 제대로 작동이 되

지 않았다. 똑 같은 방법으로 연결한 다른조들은 모두 제어가 되는데 우리조만 제어가 되지않아 어쩔 수 없이 다시 하나의 기판을 더 제작하였다. 이 기판으로 1차 컨테스트 전날 새벽까지는 센서를 제외하고는 제어가 되는 듯 하였으나 배선을 우리배에 맞게 약간 손보던 중에 갑자기 전혀 제어가 되지 않아버렸다. 다행히 1차 컨테스트는 수동으로 컨테스트를 해서 우리조가 좋은 성적으로 지나갔지만 2차 컨테스트가 일주일 밖에 남지 않아 안 되는 제어부를 어떻게 해야 할 지 걱정이 많았다. 기판을 이리 고쳐보고 저리 고쳐보고 하다가 결국 서보와 변속기를 기판에 연결할 때 신호선 뿐만 아니라 전원선과 그라운드까지 공통으로 사용하면 문제가 해결된다는 것을 깨닫고 배선을 다시 했다. 특히 우리조는 기존의 센서를 과감히 포기하고 새로운 센서를 사서 전날부터 연구를 시작해서 2차 컨테스트 전날 새벽에 센서의 사용법을 익히고 센서를 장착했으나 센서테스트용 기판에서는 잘 되던 센서가 기존의 컨트롤러 기판에서는 작동하지 않아 어디가 문제인지 파악할 시간이 부족해 그냥 센서테스트용 기판을 급히 컨트롤러 기판으로 만들어서 사용했다. 마지막 컨테스트에 시운전을 하지 못하고 나가서 제대로 된 수치를 입력하지 못해서 멋지게 가지는 못했지만 분명히 센서가 가장 정확히 작동해서 흐뭇하고 또 아쉬웠다.

계획은 정말 많이 고생하고 힘들었던 과목이 었다. 그러나 힘든만큼 정말 많은 것이 남았다. 열심히 하면 정말 안될 것 같은 일도 가까이 다다를수 있다는 것을 깨달았고 자신감을 얻었다. 또 같은 과 애들끼리 정말 꺼리낌 없이 친해 질수 있는 계기가 되었다. 힘든 일이 끝나서 좋은면도 있지만 컨테스트가 끝나고 실업자가 된 기분이 들 정도로 허전한 마음이 든다. 정말 열심히 했고 나름 좋은 결과를 얻을 수 있었던 계획 과목은 내가 대학에 와서

들었던 다른 어떤 과목보다 나에게 많은 것을 생각하게 하고 많은 것을 남겨준 과목이 되었다.



9. 후배들에게 남기는 말...

내년에 들을 후배들에게 먼저 경험한 경험자로서 해줄 말이 있어 남긴다. 원래는 조원들 마다 각각 하나씩 글을 남기기로 했으나 중복되는 말들로 인해 혼란스러울 것이라고 생각되어 하나의 글로 통일되고 일관성 있게 말하려 한다. '조선해양공학계획'이란 수업, 너희들도 선배들을 통해 익히 들어봐서 조금은 알 것이다. 재미있겠다고 기대하는 몇몇 녀석들도 있었지만 대부분 힘들 것 같다며 하기 싫어하는 마음들이었지. 우리도 처음엔 그런 마음으로 이 수업에 수강신청을 하게 되었단다. 기대 반 걱정 반으로. 하지만 모두 끝내고 나서 드는 마음은 이 수업을 듣길 잘 했다는 생각이야. 조선과 학생으로서 직접 배를 만들어 설계를 하고 만들어서 물에 띄운다는게 당연한 것이라고 생각한다. 그 이유 때문만은 아니다. 먼저 이런 team 단위로 project를 하기 위해선 협동심이 있어야 하고 조원간에 갈등을 일으키지 않으며 대화를 하는 방법도 알고 있어야 한다. 이번 학기의 예로 어떤 조는 조장과 조원간의 의사소통이 잘 되지 않아 불화가 생기는 경우가 있었고 다른 조는 조원간의 작업하는데 있어서 의견차이를 보이며 다투는 모습도 보았다. 그래서 하는 말인데 조가 짜여지고 그 조가 맘에 들지 않더라도 열심히 참여하고 상대방의 의견에 존중하는 마음가짐을 갖길 바란다. 1학년 때 진입생들 중 다른 과에서 지내다 2학년 진입 때 뜻하지 않게 조선과로 와서 흥미가 없는 사람들도 있을 줄 안다. 하지만 이 수업은 전적으로 team으로 활동하기 때문에 한명이라도 작업에 소홀하면 다른 사람들이 배로 힘들다는 것을 생각하고 함께 도와 일을 하길 바란다. 수업이 끝나면 각 조마다 개인적으로

조원들의 순위를 정해 성적에 반영을 하게 되어있다. 그렇다고 너무 성적에 연연하지는 말돼 이 부분을 상기시키면서 참여에 열심을 다하길 바란다. 지금까지 말한 것은 전체적인 활동에 임하는 자세를 말한 것이고 본격적인 project 작업에 대해서 말을 해보련다.

먼저 07년도의 project때는 어떤 주제를 삼을지는 모르겠지만 기본적으로 선체부와 제어부로 나눠진다. 일단 선체부에 대해 말해보면 처음 선형 선택이 매우 중요함을 말하고 싶다. 이번 project의 주제는 '선박제어자동화' 이기 때문에 선형은 중요하지 않다고 교수님께서 미리 말씀하셨지. 하지만 우리 조는 그 말을 듣고도 이지스함 같은 어려운 선형을 택해 교수님께 '선형 선택 전면 재수정'이란 평가를 들었지. 우리가 project에 의도를 잘 파악하지 못 했던 점이 문제가 되었던 거야. 너희들도 우리와 같은 실수를 하지 않도록 project의 주제의 의도를 먼저 파악하도록 하렴. 설계는 조교님이 EzSHIP이란 program을 통해 자세히 가르쳐주실 테니 너무 걱정하지 말아라. 하나 조심하자면 처음에 설계할 때 재료의 밀도 값을 잘 알아보고 계산해서 설계할 때 경하중량에 대한 추정을 올바르게 하고 만들어야 나중에 선체완성 후에 흘수선이 맞지 않는 문제점이 있으므로 조심하거라. 설계 후에 도면을 만들고 나서 cutting을 할 때 홈을 일일이 파면 어려우므로 칼심으로 도구를 만들면 편할 것이다. 얇은 우드락으로 외판을 두를 때 풀을 사용하면 정교하고 매끈한 선형을 만들 수 있지만 간단한 선형이라면 테이프를 이용해 쉽게 붙일 수 있다. Putty를 바를 때는 처음부터 표면을 매끄럽게 바르는 게 중요하다. Putty를 바르기 쉬우면서 빨리 마를 수 있도록 경화제를 잘 섞는 것도 중요하다. Putty 굳은 외형이 그 선체의 외형을 결정하기 때문에 바를 때 잘 바르고 말랐을 때 사포질을 열심히 해서 매끄럽게 해줘야 한다.

Putty 바르는 칼을 따로 구매해서 바르면 잘 펴 바를 수 있을 것이다. 손으로 먼저 골고루 펴 발라주고 칼로 마무리 짓는 식으로 하면 될 것이다. 2차 putty 작업 할 때는 매끄럽지 못한 부분을 손으로 살짝 발라 채우는 식으로 하면 될 것이다. 그 다음 순서로는 에폭시를 바르고 다음에 페인트를 발라 방수처리를 해야 하는데 꼼꼼히 해서 나중에 물이 새지 않도록 조심해라. 다음은 추진부를 조립하는 것인데, 공지되어 있는 재료만 산다면 조립방법을 몰라 헤매게 될 것이다. 그러니 재료 사는 곳에서 조립법을 배워서 해야 실수 없이 만들 수 있을 것이다. 이 때 추진부 부품 중에 볼베어링 같은 작은 부품들이 분실되지 않도록 조심해야 한다. 아마 선체작업에 마지막 작업이 흘수선 그리는 것이 될 텐데, 분면 너희들이 설계한대로 흘수선을 긋는다고 그 흘수선이 실제 재화중량을 싣었을 때와 비교해보면 다르다는 것을 알 것이다. 그러니 다 만들고 나서 재화중량을 싣고 그 때의 잠기는 부분을 흘수선으로 그려야 한다.

다른 part로는 제어부가 있는데, 이 부분이 매우 어려운 부분이므로 많은 시행착오를 통해 하나하나 해결해 나가야 할 것이다. 이 부분에서 예상 외로 재료비가 많이 드니 예산사용에 있어 조심해야 할 것이다. 선체부는 재료비가 어느 정도 정해져 있어 그 이상 예산이 들지 않지만 제어부는 많은 실패를 통해 재료비가 많이 들게 된다.

마지막으로 project 전반적인 조언을 하겠다. 첫 번째로는 자기 물건을 잘 챙겨야 된다는 점이다. 같은 작업실을 사용하고 비슷한 재료를 사용하기 때문에 자기 조 물건을 제대로 챙기지 못한다면 다른 조가 가져 다가 써도 찾지 못하게 되는 경우가 생기게 된다. 두 번째로는 무슨 일을 하든지 항상 기록을 남기는 점이다. 실패를 하든 성공을 하든 보고서 쓸 때 사진을 중요한

자료가 되기 때문이다. 실패를 했으면 어떻게 실패를 했는지, 그 대안으로는 어떤 방안을 냈는지, 그 후에 결과는 어땠는지를 하나하나 기록으로 남겨 보고서에 첨부하고 발표할 때 보여줘야 한다. 마지막으로는 contest 당일에 쓸 배터리를 잘 충전해 놓아야 한다. Contest 바로 전까지 시험한다고 배터리를 쓰다 보면 막상 contest 할 때에 배터리가 나가 배가 멈춰버리는 불상사가 생기게 된다. 이번 1차 contest 때만 해도 몇몇 조가 배터리가 모두 방전되어 수조 한 가운데서 멈춰 막대기로 끌어내는 상황이 발생했다. 배터리관리 소홀은 가장 문제로 보고 실격처리 하므로 꼭 조심하도록 해라.

07년도 2학기에 '조선해양공학계획'을 듣게 될 후배들에게 처음 가는 길 조금이나마 덜 실수 하고 덜 고민하기를 바라며 이 글을 남긴다.