LAMPIRAN A
PROGRAM UTAMA ROBOT NOMOR 2

```
START PROGRAM
{
    Buzzer time = Play Melody
    Buzzer index = Melody0
    ID[20]: Goal position = 330
    ID[19]: Goal position = 512
    Motion Page = 224
    Cari = 94
    Color = 1
    a = 0
    Accelero = 0
    Langkah = 0
    Session = 0
    Pan = 512
    CountCariTarget = 0
    WaitKick = 0
    Timer = 0.000sec
    Buzzer time = Play Melody
    Buzzer index = Melody0

    // ============================================================
    ENDLESS LOOP
    {
        CALL TrackBola
        CALL DigitalInput
```

```
26:             CALL KirimDataJarak
27:             IF ( Start  ==  TRUE  )
28:             {
29:                 JUMP MainProgram
30:             }
31:         }
32:         // ========================================
    ========================
33:         ENDLESS LOOP
34:         {
35:     MainProgram  :
36:             CALL TrackBola
37:             CALL DigitalInput
38:             CALL KirimDataJarak
39:             CALL Accelero
40:
41:             Aux LED  = FALSE
42:             IF ( Remocon RXD  ==  1  )
43:             {
44:                 LOOP WHILE  ( Remocon RXD  ==  1  )
45:                 {
46:                     Aux LED  = TRUE
47:                     CALL ExitExcuteStop
48:                     CALL WaitMotion
49:                     Motion Page  = 224
50:                     CALL WaitMotion
51:                     CALL TrackBola
```

```
52:                     CALL KirimDataJarak
53:                     IF ( Remocon RXD != 1 )
54:                     {
55:                         BREAK LOOP
56:                     }
57:                 }
58:             }
59:             ELSE
60:             {
61:                 Aux LED = FALSE
62:                 IF ( Tilt <= 305 && Max != 0 )
63:                 {
64:                     Remocon TXD = 1
65:                     CALL Kick
66:                 }
67:                 ELSE IF ( Tilt >= 305 && Tilt < 325 && Cc
    ariTarget < 200 )
68:                 {
69:                     LOOP WHILE ( CountCariTarget < 200 )
70:                     {
71:                         Color = 2
72:                         CALL TrackKickTarget
73:                         Remocon TXD = 1
74:                         CountCariTarget = CountCariTarget -
75:                         IF ( Pan > 488 && Pan < 550 )
76:                         {
77:                             CALL WaitMotion
```

```
78:                         CountCariTarget  = 201
79:                         Max  = 0
80:                         Color  = 1
81:                         Motion Page  = 224
82:                         CALL WaitMotion
83:                         Remocon TXD  = Tilt
84:                         ID[20]: Goal position  = 330
85:                         ID[19]: Goal position  = 512
86:                         BREAK LOOP
87:                     }
88:                     IF ( CountCariTarget  >=  200  )
89:                     {
90:                         CALL WaitMotion
91:                         CountCariTarget  = 201
92:                         Max  = 0
93:                         Color  = 1
94:                         Motion Page  = 224
95:                         CALL WaitMotion
96:                         Remocon TXD  = Tilt
97:                         ID[20]: Goal position  = 330
98:                         ID[19]: Goal position  = 512
99:                         BREAK LOOP
100:                    }
101:                }
102:            }
103:            ELSE
104:            {
```

```
105:                        CALL Walking
106:                    }
107:                }
108:
109:                IF ( Start  ==  FALSE  )
110:                {
111:                    JUMP Pause
112:                }
113:            }
114:            // ========================================
        ========================
115:            ENDLESS LOOP
116:            {
117:        Pause :
118:                Motion Page  = 224
119:                CALL DigitalInput
120:                CALL TrackBola
121:                CALL KirimDataJarak
122:                IF ( Start  ==  TRUE  )
123:                {
124:                    JUMP MainProgram
125:                }
126:            }
127:        }
128:        // ========================================
        ======================
129:        FUNCTION  ExitExcuteStop
```

```
130:    {
131:        Motion Page = 0
132:        CALL WaitMotion
133:    }
134:    // ////////////////////////////////////////////////////////////////////////////////Exit
        utExecuteStop
135:    FUNCTION ExitWithoutExcuteStop
136:    {
137:        Motion Page = -1
138:        CALL WaitMotion
139:    }
140:    // ////////////////////////////////////////////////////////////////////////////////Wai
        on
141:    FUNCTION WaitMotion
142:    {
143:        WAIT WHILE ( Motion Status == TRUE )
144:    }
145:    // ////////////////////////////////////////////////////////////////////////////////Wai
        r
146:    FUNCTION WaitTimer
147:    {
148:        WAIT WHILE ( Timer > 0.000sec )
149:    }
150:    // ////////////////////////////////////////////////////////////////////////////////GET
        ================================================================================
151:    FUNCTION Get_Bounding_Box
152:    {
```

```
153:        Max  = 0
154:        LOOP FOR ( Index  = 1  ~ 15 )
155:        {
156:            Addr  = Index  *  16
157:            IF ( ID[CamID]: ADDR[Addr(b)]  !=  0 )
158:            {
159:                Addr  = Addr  +  1
160:                IF ( ID[CamID]: ADDR[Addr(b)]  ==  Color  )
161:                {
162:                    Addr  = Addr  +  1
163:                    Size  = ID[CamID]: ADDR[Addr(w)]
164:                    IF ( Size  >  Max  )
165:                    {
166:                        Max  = Size
167:                        MaxAddr  = Addr
168:                    }
169:                }
170:            }
171:        }
172:        Addr  = MaxAddr  +  10
173:        Maxx  = ID[CamID]: ADDR[Addr(b)]
174:        Addr  = Addr  +  1
175:        Minx  = ID[CamID]: ADDR[Addr(b)]
176:        Addr  = Addr  +  1
177:        Maxy  = ID[CamID]: ADDR[Addr(b)]
178:        Addr  = Addr  +  1
179:        Miny  = ID[CamID]: ADDR[Addr(b)]
```

```
180:    }
181:    // ////////////////////////////////////////////////////////////////////////////////Trac
182:    FUNCTION  TrackBola
183:    {
184:        Color  = 1
185:        CamID  = 100
186:        ID[CamID]: ADDR[0(b)]  = 100
187:        Timer  = 0.256sec
188:        CALL WaitTimer
189:        CALL Get_Bounding_Box
190:        Cx  = Minx  +  Maxx
191:        Cy  = Miny  +  Maxy
192:        Cx  = Cx  /  2
193:        Cy  = Cy  /  2
194:
195:        IF ( Max  !=  0  )
196:        {
197:            WaitCounter  = 0
198:            Delta  = Cx  -  81
199:            Pan  = ID[19]: Present position  -  Delta
200:            Delta  = Cy  -  60
201:            Tilt  = ID[20]: Present position  -  Delta
202:
203:            IF ( Pan  >  700  )
204:                ID[19]: Goal position  = 700
205:            IF ( Pan  <  280  )
```

```
206:                ID[19]: Goal position = 280
207:            IF ( Pan >= 280 && Pan <= 700 )
208:                ID[19]: Goal position = Pan
209:
210:            IF ( Tilt < 335 )
211:                ID[20]: Goal position = 335
212:            IF ( Tilt > 512 )
213:                ID[20]: Goal position = 512
214:            IF ( Tilt < 512 && Tilt > 335 )
215:                ID[20]: Goal position = Tilt
216:
217:            Offset = Pan - 512
218:            Offset = Offset * -2
219:            IF ( Offset > 40 )
220:                Offset = 40
221:            IF ( Offset < -40 )
222:                Offset = -40
223:        }
224:        IF ( Max == 0 )
225:        {
226:            WaitCounter = WaitCounter + 1
227:            IF ( WaitCounter > 3 )
228:            {
229:                IF ( ID[20]: Present position >= 512 )
230:                {
231:                    ID[20]: Goal position = 310
232:                    IF ( ID[19]: Present position <= 300
```

```
233:                        {
234:                            ID[19]: Goal position = 695
235:                        }
236:                        IF ( ID[19]: Present position > 300 )
237:                        {
238:                            ID[19]: Goal position = ID[19]:
    esent position - 135
239:                        }
240:                    }
241:                    IF ( ID[20]: Present position < 512 )
242:                        ID[20]: Goal position = ID[20]: Pr
    t position + 87
243:                    Timer = 0.128sec
244:                    CALL WaitTimer
245:                }
246:            }
247:        }
248:        FUNCTION TrackKickTarget
249:        {
250:            ID[20]: Goal position = 498
251:            CamID = 100
252:            ID[CamID]: ADDR[0(b)] = 100
253:            Timer = 0.256sec
254:            CALL WaitTimer
255:            CALL Get_Bounding_Box
256:            Cx = Minx + Maxx
257:            Cy = Miny + Maxy
```

```
258:        Cx = Cx / 2
259:        Cy = Cy / 2
260:        IF ( Max != 0 )
261:        {
262:            WaitCounter = 0
263:            Delta = Cx - 85
264:            Pan = ID[19]: Present position - Delta
265:            Delta = Cy - 60
266:            Tilt = ID[20]: Present position - Delta
267:
268:            IF ( Pan > 700 )
269:                ID[19]: Goal position = 700
270:            IF ( Pan < 280 )
271:                ID[19]: Goal position = 280
272:            IF ( Pan >= 280 && Pan <= 700 )
273:                ID[19]: Goal position = Pan
274:
275:            Offset = Pan - 512
276:            Offset = Offset * -2
277:            IF ( Offset > 40 )
278:                Offset = 40
279:            IF ( Offset < -40 )
280:                Offset = -40
281:            // Motion :
282:            IF ( Pan <= 488 )
283:            {
284:                // Putar ke kanan
```

```
285:                    Motion Page = 231
286:                    CALL WaitMotion
287:                    // Geser ke kiri
288:                    Motion Page = 245
289:                    ID[19]: Goal position = ID[19]: Preser
    ition + 25
290:                }
291:                CALL WaitMotion
292:                IF ( Pan >= 550 )
293:                {
294:                    // Putar ke kiri
295:                    Motion Page = 232
296:                    CALL WaitMotion
297:                    // Geser ke kanan
298:                    Motion Page = 244
299:                    CALL WaitMotion
300:                    ID[19]: Goal position = ID[19]: Preser
    ition - 25
301:                }
302:            }
303:            IF ( Max == 0 )
304:            {
305:                WaitCounter = WaitCounter + 1
306:                IF ( WaitCounter > 3 )
307:                {
308:                    IF ( ID[19]: Present position <= 300 )
309:                    {
```

```
310:                        ID[19]: Goal position = 800
311:                    }
312:                    ELSE IF ( ID[19]: Present position > 300
313:                    {
314:                        ID[19]: Goal position = ID[19]: Pr
    t position - 120
315:                    }
316:                    WaitCounter = 2
317:                }
318:            }
319:        }
320:        FUNCTION Walking
321:        {
322:            IF ( Pan >= 430 && Pan <= 570 )
323:            {
324:                CALL Accelero
325:                Aux LED = FALSE
326:                IF ( Motion Page == 37 || Motion Page ==
    || Motion Page == 224 || Motion Page == 121 )
327:                {
328:                    Motion Page = 38
329:                    Step = Step + 1
330:                }
331:                IF ( Motion Page == 39 || Motion Page ==
    || Motion Page == 123 )
332:                {
333:                    Motion Page = 36
```

```
334:                }
335:            }
336:            IF ( Pan  >  570  )
337:            {
338:                Aux LED  = TRUE
339:                // FLT_M_L
340:                IF ( Motion Page  ==  109  ||  Motion Page  ==
    ||  Motion Page  ==  224  )
341:                {
342:                    Motion Page  = 110
343:                    Step  = Step  +  1
344:                }
345:                IF ( Motion Page  ==  111  ||  Motion Page  ==
    )
346:                    Motion Page  = 108
347:            }
348:            IF ( Pan  <  430  )
349:            {
350:                Aux LED  = TRUE
351:                // FRT_M_L
352:                IF ( Motion Page  ==  121  ||  Motion Page  ==
    ||  Motion Page  ==  224  )
353:                {
354:                    Motion Page  = 122
355:                    Step  = Step  +  1
356:                }
357:                IF ( Motion Page  ==  123  ||  Motion Page  ==
    )
```

```
358:                Motion Page = 120
359:            }
360:            // Jika meleset terlalu jauh
361:            IF ( Pan < 333 )
362:            {
363:                // Putar ke kanan
364:                Motion Page = 230
365:                LOOP WHILE ( Pan < 480 )
366:                {
367:                    CALL TrackBola
368:                    Motion Page = 231
369:                    ID[19]: Goal position = ID[19]: Preser
    ition + 30
370:                    CALL WaitMotion
371:                    IF ( Pan >= 480 || Max == 0 )
372:                    {
373:                        CALL WaitMotion
374:                        Motion Page = 224
375:                        BREAK LOOP
376:                    }
377:                }
378:            }
379:            IF ( Pan > 670 )
380:            {
381:                // Putar ke kiri
382:                Motion Page = 230
383:                LOOP WHILE ( Pan > 520 )
```

```
384:                {
385:                    CALL TrackBola
386:                    Motion Page = 232
387:                    ID[19]: Goal position = ID[19]: Preser
    ition - 30
388:                    CALL WaitMotion
389:                    IF ( Pan <= 520 || Max == 0 )
390:                    {
391:                        CALL WaitMotion
392:                        Motion Page = 224
393:                        BREAK LOOP
394:                    }
395:                }
396:            }
397:        }
398:        FUNCTION Kick
399:        {
400:            Motion Page = 230
401:            LOOP WHILE ( Tilt <= 320 && Max != 0 )
402:            {
403:                CALL TrackBola
404:                IF ( Pan >= 565 && Pan < 600 )
405:                {
406:                    WaitKick = WaitKick + 1
407:                    IF ( WaitKick > 10 )
408:                    {
409:                        // Kaki Kiri Tendang
```

```
410:                    CALL WaitMotion
411:                    Motion Page = 239
412:                    CALL WaitMotion
413:                    Motion Page = 224
414:                    CALL WaitMotion
415:                    WaitKick  = 0
416:                    CountCariTarget  = 0
417:                    BREAK LOOP
418:                }
419:            }
420:            IF ( Pan  >  425  &&  Pan  <=  512 )
421:            {
422:                WaitKick  = WaitKick  +  1
423:                IF ( WaitKick  >  10 )
424:                {
425:                    // Kaki Kanan Tendang
426:                    CALL WaitMotion
427:                    Motion Page = 237
428:                    CALL WaitMotion
429:                    WaitKick  = 0
430:                    CountCariTarget  = 0
431:                    Motion Page = 224
432:                    CALL WaitMotion
433:                    BREAK LOOP
434:                }
435:            }
436:            IF ( Pan  >  512  &&  Pan  <  565 )
```

```
437:        {
438:            // Prioritas Kaki Kanan Tendang
439:            // Geser Kiri
440:            Motion Page = 235
441:            CALL WaitMotion
442:        }
443:        IF ( Pan >= 600 )
444:        {
445:            // Geser Kiri
446:            LOOP WHILE ( Pan >= 600 )
447:            {
448:                Motion Page = 235
449:                CALL WaitMotion
450:                Timer = 0.128sec
451:                WAIT WHILE ( Timer > 0.000sec )
452:
453:                CALL TrackBola
454:                IF ( Pan < 560 )
455:                {
456:                    Timer = 0.256sec
457:                    WAIT WHILE ( Timer > 0.000sec
458:                }
459:            }
460:        }
461:        IF ( Pan < 425 )
462:        {
463:            // Geser Kanan
```

```
464:                    LOOP WHILE ( Pan <= 425 )
465:                    {
466:                        Motion Page = 234
467:                        CALL WaitMotion
468:                        Timer = 0.128sec
469:                        CALL WaitTimer
470:                        CALL TrackBola
471:                        IF ( Pan > 450 )
472:                        {
473:                            Timer = 0.256sec
474:                            CALL WaitTimer
475:                        }
476:                    }
477:                }
478:                Motion Page = 224
479:            }
480:        }
481:        FUNCTION Accelero
482:        {
483:
484:            IF ( PORT[3] > 410 )
485:            {
486:                // F_Getup
487:                Timer = 1.024sec
488:                CALL WaitTimer
489:                CALL ExitExcuteStop
490:                IF ( PORT[3] > 410 )
```

```
491:                {
492:                    Motion Page = 27
493:                    CALL WaitMotion
494:                    Motion Page = 224
495:                }
496:            }
497:            IF ( PORT[3] < 265 )
498:            {
499:                // B_Getup
500:                Timer = 1.024sec
501:                CALL WaitTimer
502:                IF ( PORT[3] < 265 )
503:                {
504:                    CALL ExitExcuteStop
505:                    Motion Page = 28
506:                    CALL WaitMotion
507:                    Motion Page = 224
508:                }
509:            }
510:        }
511:        FUNCTION HealthStatus
512:        {
513:            IF ( ID[14]: Temperature > 60 || ID[13]: Tem
    ure > 60 )
514:            {
515:                Buzzer time = Play Melody
516:                Buzzer index = Melody5
```

```
517:        }
518:        IF ( ID[1]: Voltage < 11 )
519:        {
520:            Buzzer time = Play Melody
521:            Buzzer index = Melody6
522:        }
523:    }
524:    FUNCTION DigitalInput
525:    {
526:        IF ( PORT[1] == TRUE )
527:        {
528:            Start = TRUE
529:        }
530:        ELSE IF ( PORT[1] == FALSE )
531:        {
532:            Start = FALSE
533:        }
534:
535:        IF ( PORT[2] == TRUE )
536:        {
537:            FirstKick = TRUE
538:        }
539:        ELSE IF ( PORT[2] == FALSE )
540:        {
541:            FirstKick = FALSE
542:        }
543:
```

```
544:            IF ( PORT[5] == TRUE )
545:            {
546:                PickUP = TRUE
547:            }
548:            ELSE IF ( PORT[5] == FALSE )
549:            {
550:                PickUP = FALSE
551:            }
552:
553:            IF ( PORT[6] == FALSE )
554:            {
555:                // Gawang Target Kuning
556:                Color = 2
557:            }
558:            ELSE IF ( PORT[6] == TRUE )
559:            {
560:                // Gawang Target Biru
561:                Color = 3
562:            }
563:        }
564:        FUNCTION KirimDataJarak
565:        {
566:            IF ( Max != 0 )
567:            {
568:                Remocon TXD = Tilt
569:            }
570:            IF ( Max == 0 )
```

```
571:        {
572:            Remocon TXD = 0
573:        }
574:        IF ( Tilt <= 300 && Max != 0 )
575:        {
576:            Remocon TXD = 1
577:        }
578:    }
```

LAMPIRAN B
PROGRAM UTAMA ROBOT NOMOR 3

```
1: START PROGRAM
2: {
3:     Buzzer time = Play Melody
4:     Buzzer index = Melody0
5:     ID[20]: Goal position = 330
6:     ID[19]: Goal position = 512
7:     Motion Page = 224
8:     Cari = 94
9:     Color = 1
10:    a = 0
11:    Accelero = 0
12:    Langkah = 0
13:    Session = 0
14:    Pan = 512
15:    CountCariTarget = 0
16:    WaitKick = 0
17:    Timer = 0.000sec
18:    Buzzer time = Play Melody
19:    Buzzer index = Melody0
20:
21:    // ============================================
   ==========================
22:    ENDLESS LOOP
23:    {
24:        CALL TrackBola
25:        Print = Pan
```

```
26:             Print with Line = Tilt
27:             // CALL DigitalInput
28:             // CALL OlahDataJarak
29:             // Motion Page = 224
30:             // IF ( Button == S || Start == TRUE )
31:             // {
32:             // JUMP MainProgram
33:             // }
34:         }
35:         // ================================================
    ==========================
36:         ENDLESS LOOP
37:         {
38:  MainProgram :
39:             CALL TrackBola
40:             CALL DigitalInput
41:             CALL OlahDataJarak
42:             CALL Accelero
43:
44:             Aux LED = FALSE
45:             IF ( Tilt <= 305 && Max != 0 )
46:             {
47:                 Remocon TXD = 1
48:                 CALL Kick
49:             }
50:             IF ( Remocon RXD < Tilt && Remocon RXD
    200 )
```

```
51:         {
52:             Remocon TXD = 1
53:         }
54:         IF ( Tilt >= 305 && Tilt < 325 && CountCariTar
    < 200 )
55:         {
56:             LOOP WHILE ( CountCariTarget < 200 )
57:             {
58:                 Color = 2
59:                 CALL TrackKickTarget
60:                 Remocon TXD = 1
61:                 CountCariTarget = CountCariTarget + 1
62:                 IF ( Pan > 488 && Pan < 550 && Ma
    0 )
63:                 {
64:                     CALL WaitMotion
65:                     CountCariTarget = 201
66:                     Color = 1
67:                     Motion Page = 224
68:                     CALL WaitMotion
69:                     Remocon TXD = Tilt
70:                     ID[20]: Goal position = 330
71:                     ID[19]: Goal position = 512
72:                     CALL TrackBola
73:                     BREAK LOOP
74:                 }
75:                 IF ( CountCariTarget >= 200 )
```

```
76:                     {
77:                         CALL WaitMotion
78:                         CountCariTarget  = 201
79:                         Color  = 1
80:                         Motion Page  = 224
81:                         CALL WaitMotion
82:                         Remocon TXD  = Tilt
83:                         ID[20]: Goal position  = 330
84:                         ID[19]: Goal position  = 512
85:                         CALL TrackBola
86:                         BREAK LOOP
87:                     }
88:                 }
89:             }
90:             ELSE
91:             {
92:                 CALL Walking
93:             }
94:             IF ( Start  ==  FALSE  )
95:             {
96:                 JUMP Pause
97:             }
98:         }
99:         // ========================================
    ==========================
100:         ENDLESS LOOP
101:         {
```

```
102: Pause :
103:         Motion Page = 224
104:         CALL DigitalInput
105:         CALL TrackBola
106:         CALL OlahDataJarak
107:         IF ( Start == TRUE )
108:         {
109:             JUMP MainProgram
110:         }
111:     }
112: }
113: // ================================================
     ======================
114: FUNCTION ExitExcuteStop
115: {
116:     Motion Page = 0
117:     CALL WaitMotion
118: }
119: // ////////////////////////////////////////////////////////////////////////////////Exit
     utExecuteStop
120: FUNCTION ExitWithoutExcuteStop
121: {
122:     Motion Page = -1
123:     CALL WaitMotion
124: }
125: // ////////////////////////////////////////////////////////////////////////////////Wai
     on
```

```
126: FUNCTION  WaitMotion
127: {
128:     WAIT WHILE  ( Motion Status  ==  TRUE  )
129: }
130: // ////////////////////////////////////////////////////////////////////////////////Wai
     r
131: FUNCTION  WaitTimer
132: {
133:     WAIT WHILE  ( Timer  >  0.000sec  )
134: }
135: // ////////////////////////////////////////////////////////////////////////////////GET
```

```
136: FUNCTION  Get_Bounding_Box
137: {
138:     Max  = 0
139:     LOOP FOR  ( Index  = 1  ~ 15  )
140:     {
141:         Addr  = Index  *  16
142:         IF  ( ID[CamID]: ADDR[Addr(b)]  !=  0  )
143:         {
144:             Addr  = Addr  +  1
145:             IF  ( ID[CamID]: ADDR[Addr(b)]  ==  Color  )
146:             {
147:                 Addr  = Addr  +  1
148:                 Size  = ID[CamID]: ADDR[Addr(w)]
149:                 IF  ( Size  >  Max  )
150:                 {
```

```
151:                         Max  = Size
152:                         MaxAddr  = Addr
153:                     }
154:                 }
155:             }
156:         }
157:         Addr  = MaxAddr  +  10
158:         Maxx  =  ID[CamID]: ADDR[Addr(b)]
159:         Addr  = Addr  +  1
160:         Minx  =  ID[CamID]: ADDR[Addr(b)]
161:         Addr  = Addr  +  1
162:         Maxy  =  ID[CamID]: ADDR[Addr(b)]
163:         Addr  = Addr  +  1
164:         Miny  =  ID[CamID]: ADDR[Addr(b)]
165: }
166: // ////////////////////////////////////////////////////////////////////////////////Trac
```

```
167: FUNCTION  TrackBola
168: {
169:         Color  = 1
170:         CamID  = 100
171:         ID[CamID]: ADDR[0(b)]  = 100
172:         Timer  = 0.256sec
173:         CALL WaitTimer
174:         CALL Get_Bounding_Box
175:         Cx  = Minx  +  Maxx
176:         Cy  = Miny  +  Maxy
```

```
177:    Cx = Cx / 2
178:    Cy = Cy / 2
179:
180:    IF ( Max != 0 )
181:    {
182:        WaitCounter = 0
183:        Delta = Cx - 81
184:        Pan = ID[19]: Present position - Delta
185:        Delta = Cy - 60
186:        Tilt = ID[20]: Present position - Delta
187:
188:        IF ( Pan > 700 )
189:            ID[19]: Goal position = 700
190:        IF ( Pan < 280 )
191:            ID[19]: Goal position = 280
192:        IF ( Pan >= 280 && Pan <= 700 )
193:            ID[19]: Goal position = Pan
194:
195:        IF ( Tilt < 335 )
196:            ID[20]: Goal position = 335
197:        IF ( Tilt > 512 )
198:            ID[20]: Goal position = 512
199:        IF ( Tilt < 512 && Tilt > 335 )
200:            ID[20]: Goal position = Tilt
201:
202:        Offset = Pan - 512
203:        Offset = Offset * -2
```

```
204:            IF ( Offset  >  40  )
205:                Offset  = 40
206:            IF ( Offset  <  -40  )
207:                Offset  = -40
208:        }
209:        IF ( Max  ==  0  )
210:        {
211:            WaitCounter  = WaitCounter  +  1
212:            IF ( WaitCounter  >=  4  )
213:            {
214:                IF ( ID[20]: Present position  >=  400  )
215:                {
216:                    ID[20]: Goal position  = 310
217:                    IF ( ID[19]: Present position  <=  350
218:                    {
219:                        ID[19]: Goal position  = 768
220:                    }
221:                    IF ( ID[19]: Present position  >  350  )
222:                    {
223:                        ID[19]: Goal position  = ID[19]:
    esent position  -  135
224:                    }
225:                }
226:                IF ( ID[20]: Present position  <  400  )
227:                    ID[20]: Goal position  = ID[20]: Pr
    t position  +  87
228:                Timer  = 0.128sec
```

```
229:                CALL WaitTimer
230:            }
231:        }
232:    }
233:    FUNCTION  TrackKickTarget
234:    {
235:        ID[20]: Goal position  = 498
236:        CamID   = 100
237:        ID[CamID]: ADDR[0(b)]   = 100
238:        Timer  = 0.256sec
239:        CALL WaitTimer
240:        CALL Get_Bounding_Box
241:        Cx  = Minx  +  Maxx
242:        Cy  = Miny  +  Maxy
243:        Cx  = Cx  /  2
244:        Cy  = Cy  /  2
245:
246:        IF ( Max  !=  0  )
247:        {
248:            WaitCounter  = 0
249:            Delta  = Cx  -  85
250:            Pan  = ID[19]: Present position  -  Delta
251:            Delta  = Cy  -  60
252:            Tilt  = ID[20]: Present position  -  Delta
253:
254:            IF ( Pan  >  700  )
255:                ID[19]: Goal position  = 700
```

```
256:            IF ( Pan  <  280  )
257:                ID[19]: Goal position  = 280
258:            IF ( Pan  >=  280  &&  Pan  <=  700  )
259:                ID[19]: Goal position  = Pan
260:
261:            Offset  = Pan  -  512
262:            Offset  = Offset  *  -2
263:            IF ( Offset  >  40  )
264:                Offset  = 40
265:            IF ( Offset  <  -40  )
266:                Offset  = -40
267:            // Motion :
268:            IF ( Pan  <=  488  )
269:            {
270:                // Putar ke kanan
271:                Motion Page  = 231
272:                CALL WaitMotion
273:                // Geser ke kiri
274:                Motion Page  = 245
275:                ID[19]: Goal position  = ID[19]: Preser
    ition  +  25
276:            }
277:            CALL WaitMotion
278:            IF ( Pan  >=  550  )
279:            {
280:                // Putar ke kiri
281:                Motion Page  = 232
```

```
282:                CALL WaitMotion
283:                // Geser ke kanan
284:                Motion Page = 244
285:                CALL WaitMotion
286:                ID[19]: Goal position = ID[19]: Preser
    ition - 25
287:            }
288:        }
289:        IF ( Max == 0 )
290:        {
291:            WaitCounter = WaitCounter + 1
292:            IF ( WaitCounter > 3 )
293:            {
294:                IF ( ID[19]: Present position <= 300 )
295:                {
296:                    ID[19]: Goal position = 800
297:                }
298:                ELSE IF ( ID[19]: Present position > 300
299:                {
300:                    ID[19]: Goal position = ID[19]: Pr
    t position - 120
301:                }
302:                WaitCounter = 2
303:            }
304:        }
305: }
306: FUNCTION Walking
```

```
307: {
308:     IF ( Pan >= 430 && Pan <= 570 )
309:     {
310:         IF ( Motion Page == 37 || Motion Page ==
    || Motion Page == 224 || Motion Page == 121 )
311:         {
312:             Motion Page = 38
313:             Step = Step + 1
314:         }
315:         IF ( Motion Page == 39 || Motion Page ==
    || Motion Page == 123 )
316:         {
317:             Motion Page = 36
318:         }
319:     }
320:     IF ( Pan > 570 )
321:     {
322:         // FLT_M_L
323:         IF ( Motion Page == 109 || Motion Page ==
    || Motion Page == 224 )
324:         {
325:             Motion Page = 110
326:             Step = Step + 1
327:         }
328:         IF ( Motion Page == 111 || Motion Page ==
    )
329:         {
```

```
330:                Motion Page = 108
331:            }
332:        }
333:        IF ( Pan < 430 && Pan >= 333 )
334:        {
335:            // FRT_M_L
336:            IF ( Motion Page == 121 || Motion Page ==
    || Motion Page == 224 )
337:            {
338:                Motion Page = 122
339:                Step = Step + 1
340:            }
341:            IF ( Motion Page == 123 || Motion Page ==
    )
342:            {
343:                Motion Page = 120
344:            }
345:        }
346:        // Jika meleset terlalu jauh
347:        IF ( Pan < 333 )
348:        {
349:            CALL WaitMotion
350:            Timer = 0.512sec
351:            CALL WaitTimer
352:            // Putar ke kanan
353:            Motion Page = 230
354:            CALL WaitMotion
```

```
355:            LOOP WHILE ( Pan < 450 )
356:            {
357:                CALL TrackBola
358:                Motion Page = 231
359:                ID[19]: Goal position = ID[19]: Preser
    ition + 30
360:                CALL WaitMotion
361:                IF ( Pan >= 450 || Max == 0 )
362:                {
363:                    CALL WaitMotion
364:                    Motion Page = 224
365:                    CALL WaitMotion
366:                    BREAK LOOP
367:                }
368:            }
369:        }
370:        IF ( Pan > 670 )
371:        {
372:            CALL WaitMotion
373:            Timer = 0.512sec
374:            CALL WaitTimer
375:            // Putar ke kiri
376:            Motion Page = 230
377:            CALL WaitMotion
378:            LOOP WHILE ( Pan > 520 )
379:            {
380:                CALL TrackBola
```

```
381:                Motion Page = 232
382:                ID[19]: Goal position = ID[19]: Preser
    ition - 30
383:                CALL WaitMotion
384:                IF ( Pan <= 520 || Max == 0 )
385:                {
386:                    CALL WaitMotion
387:                    Motion Page = 224
388:                    CALL WaitMotion
389:                    BREAK LOOP
390:                }
391:            }
392:        }
393: }
394: FUNCTION Kick
395: {
396:        Color = 1
397:        Motion Page = 230
398:        LOOP WHILE ( Tilt <= 330 && Max != 0 )
399:        {
400:            CALL TrackBola
401:            IF ( Pan >= 565 && Pan < 580 )
402:            {
403:                WaitKick = WaitKick + 1
404:                IF ( WaitKick > 10 )
405:                {
406:                    // Kaki Kiri Tendang
```

```
407:                CALL WaitMotion
408:                Motion Page = 239
409:                CALL WaitMotion
410:                WaitKick = 0
411:                CountCariTarget = 0
412:                BREAK LOOP
413:            }
414:        }
415:        ELSE IF ( Pan > 445 && Pan <= 512 )
416:        {
417:            WaitKick = WaitKick + 1
418:            IF ( WaitKick > 10 )
419:            {
420:                // Kaki Kanan Tendang
421:                CALL WaitMotion
422:                Motion Page = 237
423:                CALL WaitMotion
424:                WaitKick = 0
425:                CountCariTarget = 0
426:                BREAK LOOP
427:            }
428:        }
429:        ELSE IF ( Pan > 512 && Pan < 565 )
430:        {
431:            // Prioritas Kaki Kanan Tendang
432:            // Geser Kiri
433:            Motion Page = 235
```

```
434:            CALL WaitMotion
435:        }
436:        ELSE IF ( Pan >= 580 )
437:        {
438:            // Geser Kiri
439:            Motion Page = 235
440:            CALL WaitMotion
441:            Timer = 0.128sec
442:            WAIT WHILE ( Timer > 0.000sec )
443:            IF ( Pan < 560 )
444:            {
445:                Timer = 0.256sec
446:                WAIT WHILE ( Timer > 0.000sec )
447:            }
448:        }
449:        ELSE IF ( Pan <= 445 )
450:        {
451:            // Geser Kanan
452:            Motion Page = 234
453:            CALL WaitMotion
454:            Timer = 0.128sec
455:            CALL WaitTimer
456:            IF ( Pan > 450 )
457:            {
458:                Timer = 0.256sec
459:                CALL WaitTimer
460:            }
```

```
461:            }
462:        }
463:        Motion Page = 224
464:        CALL WaitMotion
465:    }
466:    FUNCTION  Accelero
467:    {
468:
469:        IF ( PORT[4] > 430 )
470:        {
471:            // F_Getup
472:            Timer = 1.024sec
473:            CALL WaitTimer
474:            CALL ExitExcuteStop
475:            IF ( PORT[4] > 430 )
476:            {
477:                Motion Page = 27
478:                CALL WaitMotion
479:                Motion Page = 224
480:                CALL WaitMotion
481:            }
482:            IF ( PORT[4] < 275 )
483:            {
484:                // B_Getup
485:                Timer = 1.024sec
486:                CALL WaitTimer
487:                IF ( PORT[4] < 275 )
```

```
488:                {
489:                    CALL ExitExcuteStop
490:                    Motion Page = 28
491:                    CALL WaitMotion
492:                    Motion Page = 224
493:                    CALL WaitMotion
494:                }
495:            }
496:        }
497:    }
498:    FUNCTION HealthStatus
499:    {
500:        IF ( ID[14]: Temperature > 60 || ID[13]: Tem
    ure > 60 )
501:        {
502:            Buzzer time = Play Melody
503:            Buzzer index = Melody5
504:        }
505:        IF ( ID[1]: Voltage < 11 )
506:        {
507:            Buzzer time = Play Melody
508:            Buzzer index = Melody6
509:        }
510:    }
511:    FUNCTION DigitalInput
512:    {
513:        IF ( PORT[1] == TRUE )
```

```
514:        {
515:            Start  = TRUE
516:        }
517:        ELSE IF ( PORT[1] ==  FALSE  )
518:        {
519:            Start  = FALSE
520:        }
521:
522:        IF ( PORT[2] ==  TRUE  )
523:        {
524:            FirstKick  = TRUE
525:        }
526:        ELSE IF ( PORT[2] ==  FALSE  )
527:        {
528:            FirstKick  = FALSE
529:        }
530:
531:        IF ( PORT[5] ==  TRUE  )
532:        {
533:            PickUP  = TRUE
534:        }
535:        ELSE IF ( PORT[5] ==  FALSE  )
536:        {
537:            PickUP  = FALSE
538:        }
539:
540:        IF ( PORT[6] ==  TRUE  )
```

```
541:        {
542:            // Gawang Kuning
543:            Color  = 2
544:        }
545:        ELSE IF ( PORT[6]  ==  FALSE  )
546:        {
547:            // Gawang Biru
548:            Color  = 3
549:        }
550:  }
551:  FUNCTION  OlahDataJarak
552:  {
553:        IF ( Remocon RXD  !=  0  &&  Tilt  >  310  )
554:        {
555:            LOOP WHILE  ( Remocon RXD  <  Tilt  )
556:            {
557:                Aux LED  = TRUE
558:                Remocon TXD  = 10
559:                CALL  WaitMotion
560:                Motion Page  = 224
561:                CALL  WaitMotion
562:                CALL  TrackBola
563:                IF ( Remocon RXD  >  Tilt  ||  Remocon R>
    :=  0  ||  Tilt  <=  310  )
564:                {
565:                    BREAK LOOP
566:                }
```

```
567:
    }
568:
569:        LOOP WHILE ( Remocon RXD == 1 )
570:        {
571:            Aux LED = TRUE
572:            Remocon TXD = 10
573:            CALL WaitMotion
574:            Motion Page = 224
575:            CALL WaitMotion
576:            CALL TrackBola
577:            IF ( Remocon RXD != 1 )
578:            {
579:                BREAK LOOP
580:            }
581:        }
582:    }
583: }
```

LAMPIRAN C
PROGRAM PADA ATMEGA 8

```
/*****************************************************
This program was produced by the
CodeWizardAVR V2.05.0 Professional
Automatic Program Generator
© Copyright 1998-2010 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com

Project :
Version :
Date    : 6/27/2012
Author  : NeVaDa
Company        :
Comments:


Chip type                   : ATmega8L
Program type             : Application
AVR Core Clock frequency: 7.372800 MHz
Memory model                  :  Small
External RAM size       : 0
Data Stack size         : 256
*****************************************************/

#include <mega8.h>
#include <delay.h>
#include <stdio.h>
#include <stdlib.h>

#ifndef RXB8
#define RXB8 1
#endif

#ifndef TXB8
#define TXB8 0
#endif

#ifndef UPE
#define UPE 2
#endif

#ifndef DOR
#define DOR 3
#endif

#ifndef FE
#define FE 4
#endif

#ifndef UDRE
#define UDRE 5
#endif

#ifndef RXC
#define RXC 7
#endif

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<DOR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)

//GLOBAL VARIABLES
//int test;
//int Start=0,Test;
int Mulai=0;

// USART Receiver buffer
#define RX_BUFFER_SIZE 32
char rx_buffer[RX_BUFFER_SIZE];

#if RX_BUFFER_SIZE <= 256
unsigned char rx_wr_index,rx_rd_index,rx_counter;
```

```
#else
unsigned int rx_wr_index,rx_rd_index,rx_counter;
#endif

// This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;
int udpPointer=0,z;
char udp[128];                                          //<-VARIABLE HERE

// USART Receiver interrupt service routine
interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
status=UCSRA;
data=UDR;
if (data==0x52)
   {
        udpPointer=0;
        z=1;

   }
if( z==1)
   {
        udp[udpPointer]=data;
        udpPointer++;
        if(udpPointer>=20)z=0;
   }
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
   {
   rx_buffer[rx_wr_index++]=data;
#if RX_BUFFER_SIZE == 256
   // special case for receiver buffer size=256
   if (++rx_counter == 0)
      {
#else
   if (rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
   if (++rx_counter == RX_BUFFER_SIZE)
      {
      rx_counter=0;
#endif
      rx_buffer_overflow=1;
      }
   }
}

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index++];
#if RX_BUFFER_SIZE != 256
if (rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#endif
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
#pragma used-
#endif

// Standard Input/Output functions
#include <stdio.h>

// Declare your global variables here

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

```
// Input/Output Ports initialization
// 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
// Func6=In Func5=In Func4=In Func3=Out Func2=Out Func1=Out Func0=Out
// State6=T State5=T State4=T State3=0 State2=0 State1=0 State0=0
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=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
TCCR0=0x00;
TCNT0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: Timer1 Stopped
// Mode: Normal top=0xFFFF
// OC1A output: Discon.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 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: Timer2 Stopped
// Mode: Normal top=0xFF
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

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

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

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 115200
UCSRA=0x00;
```

```
UCSRB=0x98;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x03;

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

// ADC initialization
// ADC disabled
ADCSRA=0x00;

// SPI initialization
// SPI disabled
SPCR=0x00;

// TWI initialization
// TWI disabled
TWCR=0x00;

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

while (1)
      {
       //start(03)
       if (udp[9]==0x03 && Mulai==1)PORTC.1=1;
       if (udp[9]==0x03 && Mulai==0)PORTC.1=0;
       if (udp[9]==0x02)PORTC.1=0;
       if (udp[9]==0x01)PORTC.1=0;
       if (udp[9]==0x03)PORTC.1=0;


       //Periksa tim 03 yang akan kick off atau tidak
       if (udp[11]==0x03)PORTC.2=1;
       else PORTC.2=0;


       //Penalti untuk Robot 2
       if (udp[20]==0x07) //periksa status untuk tim 07 atau yang lain?
            {
             if (udp[21]==0x01)PORTC.3=1; //Warna Tim Magenta
             if (udp[21]==0x00)PORTC.3=0; //Warna Tim Cyan

             if (udp[22]==0x01)PORTC.4=1; //Gawang Kuning
             if (udp[22]==0x00)PORTC.4=0; //Gawang Biru

             if (udp[30]!=0x00) Mulai=0; //terjadi penalti (robot out)
             if (udp[30]==0x00) Mulai=1; // countdown penalti selesai
            }

       if (udp[68]==0x07) //periksa status untuk tim 07 atau yang lain?
            {
             if (udp[69]==0x01)PORTC.3=1; //Warna Tim Magenta
             if (udp[69]==0x00)PORTC.3=0; //Warna Tim Cyan

             if (udp[70]==0x01)PORTC.4=1; //Gawang Kuning
             if (udp[70]==0x00)PORTC.4=0; //Gawang Biru

             if (udp[78]!=0x00) Mulai=0; //terjadi penalti (robot out)
             if (udp[78]==0x00) Mulai=1; // countdown penalti selesai
            }

       //Penalti untuk Robot 3
//       if (udp[20]==0x07) //periksa status untuk tim 07 atau yang lain?
//            {
//             if (udp[21]==0x01)PORTC.3=1; //Warna Tim Magenta
//             if (udp[21]==0x00)PORTC.3=0; //Warna Tim Cyan
//
//             if (udp[22]==0x01)PORTC.4=1; //Gawang Kuning
```

```
//              if (udp[22]==0x00)PORTC.4=0; //Gawang Biru
//
//              if (udp[34]!=0x00) Mulai=0; //terjadi penalti (robot out)
//              if (udp[34]==0x00) Mulai=1; // countdown penalti selesai
//             }
//
//        if (udp[68]==0x07) //periksa status untuk tim 07 atau yang lain?
//             {
//              if (udp[69]==0x01)PORTC.3=1; //Warna Tim Magenta
//              if (udp[69]==0x00)PORTC.3=0; //Warna Tim Cyan
//
//              if (udp[70]==0x01)PORTC.4=1; //Gawang Kuning
//              if (udp[70]==0x00)PORTC.4=0; //Gawang Biru
//
//              if (udp[82]!=0x00) Mulai=0; //terjadi penalti (robot out)
//              if (udp[82]==0x00) Mulai=1; // countdown penalti selesai
//             }


       // Test=0x03 ;



       udp[udpPointer] = 0x00;
       // delay_ms(100);


       }
}
```

LAMPIRAN D

STRUKTUR DATA SOFTWARE REFEREE BOX

PENJELASAN TENTANG
STRUKTUR DATA DAN IMPLEMENTASI GAME CONTROLLER
PADA ROBO SOCCER HUMANOID LEAGUE 2012
KRCI 2012

A. KONFIGURASI SISTEM WASIT MENGGUNAKAN REFEREE BOX

Sesuai dengan aturan pertandingan Robo Soccer Humanoid League (RSHL) bahwa setiap robot wajib berkomunikasi dengan sebuah sistem perangkat pertandingan atau yang dikenal dengan Referee Box. Kegagalan atas penerimaan perintah dari Referee Box ini akan menyebabkan robot dikenai penalty waktu dengan dikeluarkan dari lapangan selama 30 detik. Sistem ini berfungsi seperti wasit yang memimpin pertandingan sepak bola. Sistem ini akan menentukan kapan sebuah pertandingan dimulai dan diakhiri. Selain ini refere box, juga menentukan kapan sebuah kejadian tendangan penalty, lemparan ke dalam (dilakukan oleh wasit), tendangan bebas harus dilakukan. "MULAI" artinya robot boleh mengejar bola. "STOP" artinya robot tidak boleh mengejar bola, dan hanya boleh berhenti atau jalan di tempat, atau mengambil posisi sesuai dengan fungsi bertahan (jika lawan melakukan tendangan bebas). Semua aktifitas robot (ketika harus STOP) ini harus dapat dilakukan secara autonomous. Setiap kali robot tidak mampu mengikuti perintah dari Referee Box (melalui wifi) ketika diperintahkan untuk MULAI atau STOP maka robot dikeluarkan dari lapangan selama 30 detik dan boleh masuk kembali dari sisi lapangan. Sisi lapangan pemain akan ditentukan sebelum pertandingan dimulai.

Pada sebuah pertandingan seluruh robot yang bertanding diwajibkan untuk mendengarkan informasi yang diberikan oleh Referee Box. Informasi ini berupa data yang dikirim melalui jalur komunikasi Wi-Fi. Dalam hal ini panitia pertandingan akan menyediakan sebuah perangkat *Access Point* yang digunakan untuk mem-*broadcast* data yang dikirimkan oleh Referee Box. Data yang dikirimkan memiliki format dan definisi yang telah ditentukan.

Secara umum konfigurasi perangkat yang akan berkerja pada sistem pertandingan liga humanoid robot soccer tampak pada gambar 1.0. pada gambar 1.0 tampak sebuah acsess point dan controller yang dilengkapi dengan modul I/O komunikasi Wi-Fi. Access point disiapkan dan berada pada sisi panitia. Sedangkan controller adalah perangkat yang harus diadakan pada sebuah robot yang bertanding agar bisa berhubungan dengan akses point sistem Referee Box.

Gambar 1.0 berikut ini adalah ilustrasi tentang implementasi komunikasi antara robot dengan Referee Box pertandingan. Pada kasus ini controller yang digunakan pada robot adalah merk RABBIT type RCM 4300. Controller ini dilengkapi dengan editor program dan debugger dengan bahasa pemrogramman Dynamic C.

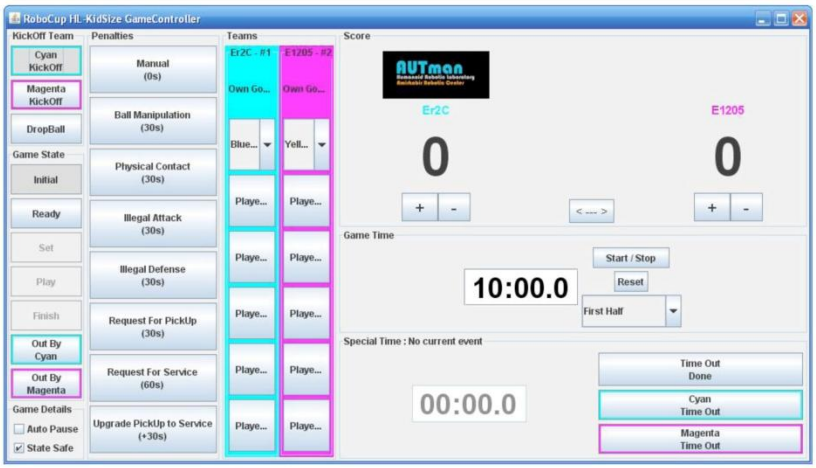
Gambar 1.0: Konfigurasi sistem Referee Box Robo Soccer Humanoid League

B. REFEREE BOX

Untuk mengirimkan perintah kepada robot, wasit menggunakan program bantuan yaitu ***Referee Box***. ***Referee box*** merupakan sebuah program *PC* yang dirancang untuk menyediakan antarmuka *GUI* yang dioperasikan oleh Asisten Wasit yang berfungsi untuk menyiarkan tanpa kabel (broadcast) data-data (serial) instruksi bermain untuk kedua tim yang sedang bertanding. Selain itu ***Referee box*** ini juga dapat menyimpan data sebuah pertandingan berupa waktu pertandingan, skor, hukuman, dsb.

Referee Box yang dioperasikan dari tepi lapangan pada sebuah *laptop / PC* ini akan dijalankan hanya pada saat *Restart (mulai bertanding)* dan *Stoppages (pertandingan selesai).* Asisten wasit tidak akan melakukan apa-apa (Referee Box tidak melakukan broadcast data) pada saat pertandingan berlangsung kecuali pada saat usai *half time* dan atau *full time*.

Tampilan Referee Box ditunjukkan dalam Gambar 1.1 berikut ini.



Gambar 1.1: Referee Box

Sumber : Rule Book Robocup Humanoid league 2011

Fungsi masing-masing tombol dalam Gambar 1.1 di atas ditunjukkan dalam Tabel 1.1 berikut ini.

Tabel 1.1: *Referee Box Buttons*

Game Button	Keterangan
Cyan/Magenta Kick off	Untuk memilih tim manakah yang akan melakukan *kick off*
Drop Ball	Untuk memilih tim manakah yang akan melakukan *Drop Ball*
Initial	*Referee Box* akan memberikan kondisi kepada kedua tim, kondisi ini berisi informasi tentang *kick off* dan warna tim
Ready	Robot bersiap di lapangan pada wilayahnya masing-masing. Robot yang melakukan tendangan *kick off* berada di tengah lingkaran *kick off* dan lawannya berada di luar lingkaran *kick off* sampai *kick off* dimulai.
Set	Wasit akan menggunakan perintah ini untuk memerintahkan robot kembali ke posisi yang seharusnya jika ada robot yang berada di posisi yang ilegal. Apabila robot tidak dapat melakukannya sendiri dan ditempatkan secara manual, maka tim yang bersangkutan akan dikenakan pelanggaran. Bola tetap berada di tengah lapangan
Play	Pertandingan dimulai, robot akan memulai pertandingan dengan strategi

	masing-masing.
Finish	Pertandingan selesai
Out By Cyan	Tim *Cyan* mengeluarkan bola dari lapangan
Out By Magenta	Tim *Magenta* mengeluarkan bola dari lapangan

C. KONFIGURASI UNTUK PROGRAM RECEIVER

Berikut adalah prosedur yang harus dilakukan oleh setiap tim dalam mengaktifkan penerima wifi di robotnya agar dapat memantau perintah wasit yang dipancarkan melalui Referee Box:

1. Melakukan Konfigurasi UDP (User Datagram Protocol) untuk mode komunikasi datanya.
   Contoh:
   ```
   #define TCPCONFIG        5 //  ← Konfigurasi UDP
   ```

2. Melakukan konfigurasi untuk SSID (Service Set Identifier) yang sesuai dengan konfigurasi SSID dari Referee Box.
   [**Dalam kontes nanti Dewan Juri akan menggunakan SSID = HumanoidLeague**]
   Contoh:
   ```
   #define _WIFI_SSID       "HumanoidLeague" // ← Konfigurasi SSID
   ```

3. Menentukan IP Address, gateway, nameserver dan tipenya sesuai dengan arahan Juri.
   Contoh:
   ```
   #define _PRIMARY_STATIC_IP  "10.10.6.100"
   #define _PRIMARY_NETMASK   "255.255.255.0"
   #define MY_GATEWAY        "10.10.6.1"
   #define MY_NAMESERVER      "10.10.6.1"
   ```

   Penggunaan IP Address, gateway dan nameserver ini akan ditentukan **pada saat Technical Meeting**. Namun demikian peserta dapat melakukan ujicoba sendiri sesuai dengan perangkat Access Point yang dimilikinya dengan IP Address berbasis DHCP ataupun static

4. Mengkonfigurasi UDP pada port 3838.
   Contoh:
   ```
   #define LOCAL_PORT    3838  // ← Konfigurasi Port
   ```

   Contoh konfigurasi komplit pada Modul Wifi Rabbit RCM4400W :
   ```
   #define TCPCONFIG        5 //  ← Konfigurasi UDP
   #define _PRIMARY_STATIC_IP  "10.10.6.100"
   #define _PRIMARY_NETMASK   "255.255.255.0"
   #define MY_GATEWAY        "10.10.6.1"
   #define MY_NAMESERVER      "10.10.6.1"
   #define _WIFI_SSID       "HumanoidLeague" // ← Konfigurasi SSID
   #define _WIFI_MODE        WIFICONF_INFRASTRUCT
   #define _WIFI_REGION_REQ   _AMERICAS_REGION
   #define _WIFI_WEP_FLAG     WIFICONF_WEP_DISABLE
   #define LOCAL_PORT    3838  // ← Konfigurasi Port
   ```

Catatan:
Silakan dipelajari secara mendalam resource Game Controller daro Humanoid League Robocup.org

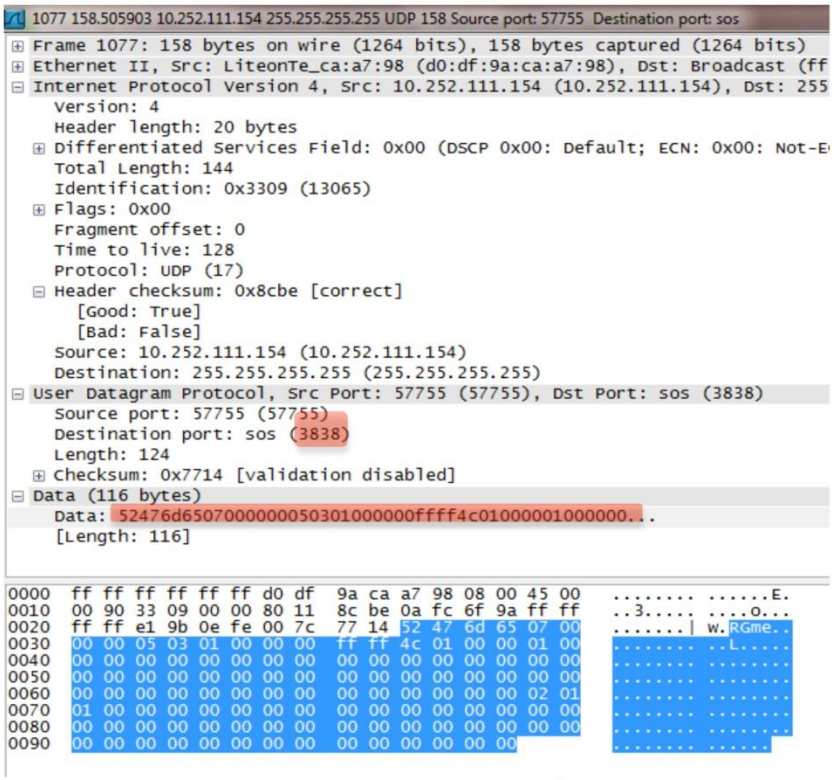
D. PAKET DATA REFEREE BOX

Berikut adalah listing program untuk paket data yang didefinisikan di source program Game Controller. Dari struct (struktur data) yang dipakai ini akan kelihatan bagaimana format data keseluruhan yang dikirimkan oleh Game Controller melalui Referee Box dalam satu frame pengiriman.

```
struct RobotInfo {
  uint16 penalty;          // penalty state of the player
  uint16 secsTillUnpenalised; // estimate of time till unpenalised
};

struct TeamInfo {
  uint8 teamNumber;        // unique team number
  uint8 teamColour;        // colour of the team
  uint8 goalColour;        // colour of the goal
  uint8 score;             // team's score
  RobotInfo players[MAX_NUM_PLAYERS];      // the team's players
};

struct RoboCupGameControlData {
  char  header[4];         // header to identify the structure
  uint32 version;          // version of the data structure
  uint8 playersPerTeam;      // The number of players on a team
  uint8 state;             // state of the game (STATE_READY, STATE_PLAYING, etc)
  uint8 firstHalf;         // 1 = game in first half, 0 otherwise
  uint8 kickOffTeam;       // the next team to kick off
  uint8 secondaryState;      // Extra state information - (STATE2_NORMAL,
STATE2_PENALTYSHOOT,etc)
  uint8 dropInTeam;        // team that caused last drop in
  uint16 dropInTime;       // number of seconds passed since the last drop in.  -1 before
first dropin
  uint32 secsRemaining;      // estimate of number of seconds remaining in the half
  TeamInfo teams[2];
};
struct RoboCupGameControlReturnData {
  char   header[4];
  uint32  version;
  uint16  team;
  uint16  player;          // player number - 1 based
  uint32  message;
};
```

E. MONITORING DATA PAKET MELALUI PROGRAM WIRESHARK

Untuk menguji struktur data UDP dari Referee Box – jika Anda membuat percobaan sendiri dengan perangkat Access Point - dapat menggunakan program WIRESHARK (dapat diunduh gratis dari internet) seperti nampak dalam Gambar 1.2 berikut ini.



Gambar 1.2. Struktur Data UDP dari Referee Box

Dari Gambar 1.2 di atas, pada tampilan yang di-blok warna merah, adalah nomer port (3838) dan data yang dikirim dalam tiap frame pengiriman. Jika data diurai nampak nilai heksa dari seluruh byte yang dikirim. Tugas robot adalah mampu menangkap seluruh data ini (terutama data untuk MULAI dan STOP) dengan menyesuaikan byte keberapa yang seharusnya diterjemahkan.

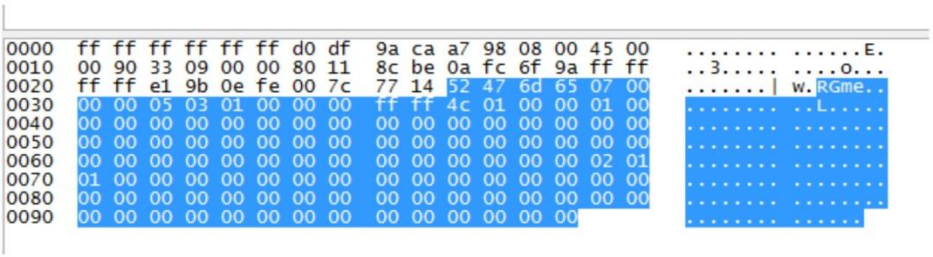
Baris modul program berikut ini adalah contoh program yang telah diujicoba untuk robot yang menggunakan modul wifi tipe Rabbit.

```
//Contoh Program :

typedef unsigned char  uint8;
typedef unsigned short uint16;
typedef unsigned int   uint32;
////////////// Fungsi Program Pengambilan Data UDP dari Referee Box Pada Rabbit
RCM4400//////////
_udpsrv_debug int receive_packet(void)
{
        static char buf[128];
        struct RoboCupGameControlData* tes;
        struct TeamInfo tim_info;
            #GLOBAL_INIT
```

```
                {
                        memset(buf, 0, sizeof(buf));
                }
                /* receive the packet */
                if (-1 == udp_recv(&sock, buf, sizeof(buf))) {
                        return 0;
                }
          tes=(RoboCupGameControlData*)buf;
        //Game State = tes->state;
        //Kickoff Team = tes->kickOffTeam
        //Sisa Waktu = tes->secsRemaining;
          return 1;
}


////////Fungsi Main Program Pada Rabbit RCM4400W/////////
void main(){

        static unsigned char val,mth,last_val,ready;
        static int i;
        static char buf,h,buff[3];

        brdInit();

        sock_init_or_exit(1);

        if(!udp_open(&sock, LOCAL_PORT, resolve(REMOTE_IP), 0, NULL)) {
                printf("udp_open failed!\n");
                exit(0);
        }
        val=0;
        while(1){
                tcp_tick(NULL);
                receive_packet();

        }
}
```

F. SEGMENTASI DATA DARI REFEREE BOX

Gambar 1.3 berikut adalah cukilan gambar data dari Gambar 1.2 di atas. Dari gambar ini diuraikan maksud dari masing-masing byte seperti yang dijelaskan dalam Tabel 1.2.



Gambar 1.3: Contoh Struktur Data dari Referee Box

Dari gambar hasil pengambilan data, ditunjukkan panjang data yang di kirim oleh referee box sebanyak 116 byte dengan rincian sebagai berikut:

Urutan Data	Info Data	Nama Data	Panjang Data (byte)	Data	Arti Data
1	Kontrol Data	Header	4	52 47 6d 65	RGme(Header)
2		Version	4	07 00 00 00	Referee box versi ke 7
3		Jumlah Pemain	1	05	Jumlah pemain 5 robot
4		State Permainan (Initial, Ready, Set, Play, Finish)	1	03	Menunjukkan dalam kondisi “PLAY”
5		Indikator Babak Pertama	1	01	Permainan dalam fase babak pertama
6		Tim yang akan kick off	1	00	Tim 0 yang akan Kick Off
7		Informasi State Tambahan (Normal ,Tendangan Pinalti ,dll)	1	00	State Normal
8		Tim Yang penyebab Drop IN	1	00	Tidak terjadi Drop In
9		Perhitungan Waktu Drop IN	2	FF FF	Tidak Terjadi Drop In
10		Perhitungan mundur waktu permainan per fase	4	4c 01 00 00	Waktu permainan 332 detik lagi
11	Info Untuk Tim 1	Nomor Tim	1	01	Tim Nomor 1, banyak nomor tergantung banyaknya tim yang ikut, nomor tiap tim berbeda.
12		Warna Tim	1	00	Warna Cyan
13		Warna Gawang	1	00	Gawang Biru
14		Skor Tim	1	00	Skor Tim = 0
15	Info Robot 1	Kondisi Pinalty(Pelanggaran, manipulasi bola, serangan illegal, pertahanan illegal, dll)	2	00 00	Tidak Terjadi Pinalti
16		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
17	Info Robot 2	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
18		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
19	Info Robot 3	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
20		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
21	Info Robot 4	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
22		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
23	Info Robot5	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
20		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
21	Info Robot 6	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
22		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
23	Info Robot 7	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
24		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
25	Info Robot 8	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
26		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
27	Info Robot 9	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
28		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti

29	Info Robot 10	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
30		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
31	Info Robot 11	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
32		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
33	Info Untuk Tim 2	Nomor Tim	1	01	Tim Nomor 2
34		Warna Tim	1	01	Warna Magenta
35		Warna Gawang	1	01	Gawang Kuning
36		Skor Tim	1	00	Skor Tim = 0
37	Info Robot 1	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
38		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
39	Info Robot 2	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
40		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
41	Info Robot 3	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
42		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
43	Info Robot 4	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
44		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
45	Info Robot5	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
46		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
47	Info Robot 6	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
48		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
49	Info Robot 7	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
50		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
51	Info Robot 8	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
52		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
53	Info Robot 9	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
54		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
55	Info Robot 10	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
56		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti
57	Info Robot 11	Kondisi Pinalty	2	00 00	Tidak Terjadi Pinalti
58		Waktu Pinalty	2	00 00	Tidak Terjadi Pinalti

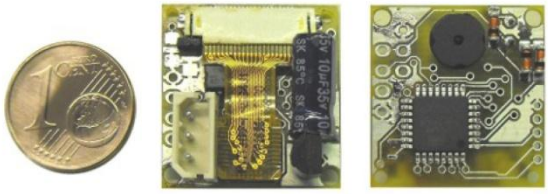
LAMPIRAN E

DATASHEET KAMERA HAVIMO

HaViMo2

Image Processing Module

March 13, 2010



Features

- Integrated Color CMOS Camera
  - Frame Resolution: 160*120 Pixels
  - Color Depth: 12 bits YCrCb
  - Frame Rate: 19 Fps
  - Full Access to all CMOS Camera registers
    - Saving values in EEPROM, no need to reconfigure after power on
    - Auto / Manual Exposure, Gain and White balance
    - Adjustable Hue/Saturation
- Color-Based Image Processing
  - Integrated Color Look-up Table
    - Saved in FLASH, No need to recalibrate after power on
    - Up to 256 Objects can be defined
    - 3D viewing and editing tools
    - Real-time LUT overlay on the Camera Image
  - On-line Region-growing
    - Detection of up to 15 contiguous Regions per Frame

    - Reporting Color, CoM, Number of Pixels and Bounding box for each region
    - Adjustable Noise / small Region filtering
  - On-line Griding
    - Reduces the Resolution of the Image to 32*24
    - Minimum Loss of Information using Object Priority
    - Reports Color and Number of Pixels for each 5x5 Cell
  - Raw image output in both calibration and implementation modes
    - Interlaced Output at 19 FPS
    - Full Frame Output at 0.5 FPS

- Supported Hardware
  - Half Duplex (*ROBOTIS*)
    - CM5
    - CM510
    - USB2Dynamixel
  - Full Duplex (*RoboBuilder*)
    - RBC
  - Requirements for other platforms
    - TTL level RS232
    - 115200 BAUD for Full Duplex Mode (*RoboBuilder* Mode)
    - 1 MBAUD for Half Duplex Mode (*ROBOTIS* Mode)

- Supported Software
  - Roboplus (*ROBOTIS*)
  - Direct C programming (*ROBOTIS*)
  - RBC firmware
  - Direct C programming (*RoboBuilder*)

1 Introduction

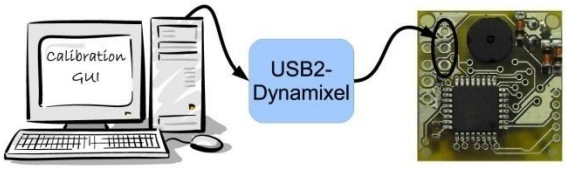
HaViMo is a computer vision solution for low power microprocessors. It is equipped with a CMOS camera chip and a microcontroller which performs the image processing. The results are then accessed via serial port. In HaViMo2 several features such as frame rate are improved.

In addition to the region growing algorithm available in prior versions, a new algorithm is implemented called *Griding*. The algorithm is a suitable pre-processing step for many other applications such as object recognition and self localization.

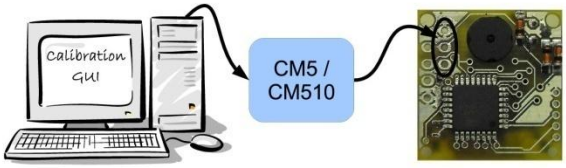
The compatibility area of the module is also extended to more software and hardware platforms. HaViMo2 is compatible with *ROBOTIS* and *RoboBuilder* bridges. It can also be integrated in programs developed in Roboplus. Other platforms can also communicate with the module using either half or full-duplex serial protocols.

2 Hardware Setup

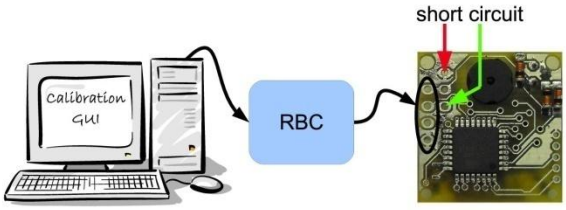
The module can be used in different configurations according to the hardware platform it is used in conjunction with. These are described in calibration and implementation modes.



a) Direct Calibration using USB2Dynamixel



b) In-System Calibration in *ROBOTIS* Mode



c) In-System Calibration in *RoboBuilder* Mode

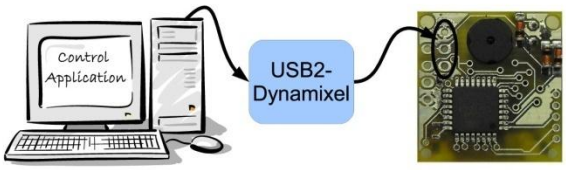
Figure 1. Possible Hardware Configurations in Calibration Mode.

In calibration mode the module is connected to a PC where a GUI facilitates the access to camera parameters as well as the color look-up table. in this mode, the camera chip can be configured and color to object associations are established by user according to the lighting conditions.

Possible configurations in calibration modes are:

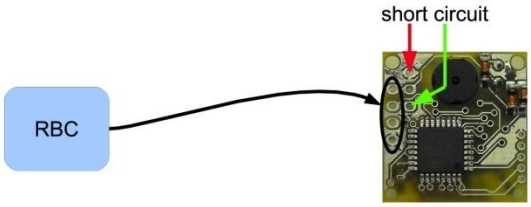
- Direct Calibration using USB2Dynamixel
- The module is connected through USB2Dynamixel to a PC. Note that it is necessary to connect the power supply externally to the device.
- In-System Calibration in *ROBOTIS* Mode
  - The module is connected through a standard *ROBOTIS* bridge (CM5, CM510) to a PC. The firmware of the bridge allows data communication between HaViMo2 and the PC

- In-System Calibration in *RoboBuilder* Mode
  - The module is connected through a standard *RoboBuilder* bridge (RBC) to a PC. The firmware of the bridge allows data communication between HaViMo2 and the PC. To enter this mode the marked pins on the PCB should be jumpered prior to module start-up.



a) Implementation on a PC Platform using USB2Dynamixel

b) Implementation in *ROBOTIS* Mode



c) Implementation in *RoboBuilder* Mode

Figure 2. Possible Hardware Configurations in Implementation Mode

3 Function Description

HaViMo2 is accessed on a serial bus. A communication protocol is designed for accessing the device which works on a command/response basis. A command packet contains an instruction which invokes a function of the device, reads or writes values or a combinations of these. In this section the function of the device is described.

3.1 Communication Protocol

HaViMo2 supports both half and full duplex communications in physical layer. This makes the module compatible with both *ROBOTIS* and *RoboBuilder* platforms. However to avoid conflicts

with other members of the bus, it is also necessary to support the communication protocols of both platforms. Following diagram shows a summary of command and response packets in both operation modes.

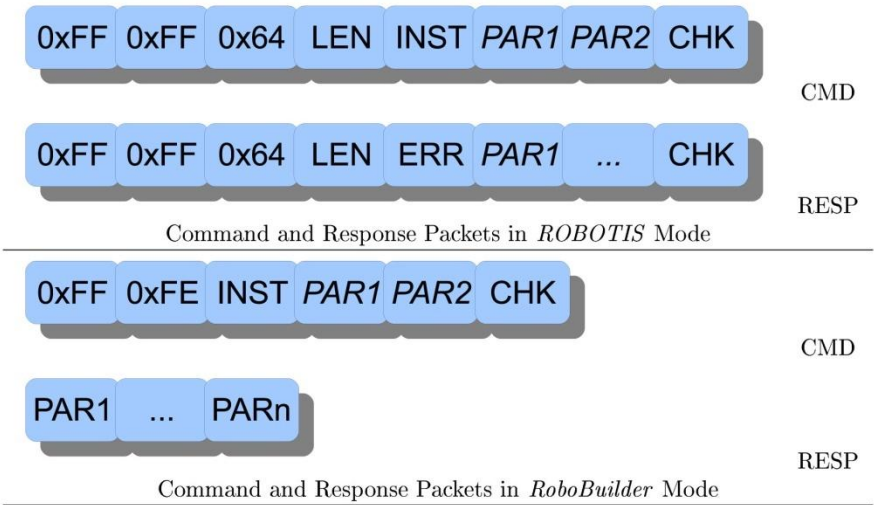


Figure 3. Command and Response Packets in *ROBOTIS* and *RoboBuilder* Modes

3.1.1 Communication Protocol in *ROBOTIS* Mode

To understand the communication protocol in *ROBOTIS* mode, it is recommended to read *Dynamixel AX-12* data sheet. The Command packet is structured as follows.

Header 2 times 0xFF.

ID Fixed on 0x64 = 100.

LEN Number of bytes to be further transmitted.

INST Instruction code described in table 1.

PARn Optional parameters passed to the instruction.

CHK Check sum is calculated as complement of the lowest 8 bits of the sum of all bytes in the packet excluding the header.

The response packet has a similar structure as the command packet, however the instruction is replaced with an error indicator and parameters are filled with the results of running the instruction.

3.1.2 Communication Protocol in *RoboBuilder* Mode

To understand the communication protocol in *RoboBuilder* mode, it is recommended to refer to the *RoboBuilder wCK* data sheet. The idea behind *RoboBuilder* compatibility mode is to simulate

a *RoboBuilder* configuration packet being transmitted to ID = 30 which is not existing on the bus, so that it is ignored by the other bus members.

The command packet is structured as follows.

Header The sequence 0xFF 0xFE, which also includes the fixed ID = 30

INST Instruction code described in table 1.

PAR1,PAR2 Parameters associated with the instruction. Note that The number of parameters MUST always be 2

CHK Check sum is the lowest 7 bits of the exclusive or of the instruction and the parameters.

3.2 Instructions

Following table shows available instruction in HaViMo2.

Instruction	Value	No. of Param.	Function
PING	0x01	0	No action. Used for obtaining a Status Packet
READ_REGION	0x02	2	Read Results of Region Detection
WRITE	0x03	2	Equivalent to CAP_REGION for Compatibility
READ_REG	0x0C	2	Read Camera Chip Registers
WRITE_REG	0x0D	2	Write Camera Chip Registers (1)
CAP_REGION	0x0E	0	Capture and Find Color Regions (1)
RAW_SAMPLE	0x0F	0	Sample the Raw Image (used by GUI) (2)
LUT_MANAGE	0x10	0	Enter LUT Manage Mode (used by GUI) (2)
RD_FILTHR	0x11	2	Read Noise Filter Thresholds
WR_FILTHR	0x12	2	Write Noise Filter Thresholds (1)
RD_REGTHR	0x13	2	Read Region Filter Thresholds
WR_REGTHR	0x14	2	Write Region Filter Thresholds (1)
CAP_GRID	0x15	0	Capture and Compress using Griding algorithm (1)
READ_GRID	0x16	2	Read Results of the Griding Algorithm (3)
SAMPLE_FAST	0x17	0	Fast Sample (used by GUI) (2) (4)

(1) No return packets are generated for these instructions.
(2) Response is different from *ROBOTIS*/*RoboBuilder* standard packets.
(3) The given address is internally multiplied by 16
(4) Not supported in *RoboBuilder* Mode

Table 1. Available Instruction in HaViMo2

PING This instruction is used to check whether the device exists and is ready to receive the next instruction. The instruction returns an empty status packet.

READ_REGION This instruction is used to read the results of the region growing algorithm. This command accepts multi byte read. The data structure is described further in the data sheet.

WRITE This instruction is to achieve compatibility to Roboplus as it only supports *READ* and *WRITE* instructions. Therefore a write to an arbitrary address simulates a *CAP_REGION* instruction.

READ_REG This instruction is to read the content of camera registers. It is the same as *READ* instruction. This command accepts multi byte read.

WRITE_REG This instruction is to write the content of camera registers. It is the same as the *WRITE* instruction but it accepts only single byte write.

CAP_REGION This instruction starts capturing and processing of the next available frame. The processing algorithm used in this instruction is *Region Growing*. It takes approximately 60 ms to process a full frame. The main CPU should pole the functionality of the device using the *PING* command before sending the next instruction. The results can be then accessed using *READ_REGION* instruction.

RAW_SAMPLE With this instruction the camera module transmits a full frame of raw image data. This instruction is used when the GUI receives a request to sample a raw image. This instruction does not use the Standard packet protocol.

LUT_MANAGE After receiving this instruction, the module enters the programming mode, in this mode the device accepts no more packets, but other instructions assigned to manage the look-up table, such as erasing, reading and writing into it. This instruction is used by User interface during calibration phase and should not be used in implementation phase.

RD_FILTHR, WR_FILTHR These instructions make access to the threshold values of the noise filter. Noise filter thresholds are actually the minimum number of neighbor pixels in a scan line which should be counted as a part of a region. For each color a separate 8-bit threshold should be defined. Default values are 8 for unknown color and 2 for others. The address field contains the index of the color category (0 = unknown,...). The structure of these instructions are the same as in *READ* and *WRITE*, however multi-byte read/write is not supported.

RD_REGTHR WR_REGTHR These instructions provide access to the threshold values of the region filter. The region filter thresholds define the minimum number of pixels inside a region, regions with fewer pixels are filtered away. For each color a separate 16-bit threshold should be defined. The address field contains 2*index of the color category (0 = unknown,...). Default values are (0,5,50,50,50,100,50,50). No region is built for color 0 (unknown). The structure of these instructions are the same as in *READ* and *WRITE*, however multi-byte read/write is not supported.

CAP_GRID This instruction invokes the griding algorithm. The algorithm compresses the image into a 32*24 cell grid and reports the number of pixels and the color observed in the 5x5 window related to the cell. Only one color is accepted for each cell. Lower color codes dominate the higher ones but Unknown = 0 has the lowest priority.

READ_GRID This instruction reads the results of the griding algorithm. It has a similar structure to other READ instructions but the given address is internally multiplied by 16. This gives access to a wider range of addresses, however at least 16 bytes should be read to cover the whole space. The data structure is described further in the data sheet.

SAMPLE_FAST This instruction is used to download a RAW image from the module. Unlike *RAW_SAMPLE* instruction, the image is transferred with full baud rate, 1Mbps. Therefore it is not possible to use the instruction with low baud rates such as in *RoboBuilder* mode.

3.3 Image Processing Algorithms

HaViMo2 is equipped with two image processing algorithms, which are described in this section. In the first step both algorithms translate color values to object codes using the built-in look-up table. Therefore an exact calibration of the colors should have a great impact on the results of the recognition.

3.3.1 On-line Region Growing Algorithm

The goal of the region growing algorithm is to detect contiguous color blobs in the image. A 4-pixel neighborhood is used to determine connections. The function is invoked using the instruction *CAP_REGION*. The detected regions are summarized using the following parameters:

Value	Bytes	Description
Index	1	Contains zero if the region is invalid and nonzero otherwise.
Color	1	Color code of the detected region (0 = Unknown, 1 = Ball , ...)
Pixels	2	Number of detected pixels inside the region
SumX	4	Sum of the X coordinates of the detected pixels (*)
SumY	4	Sum of the Y coordinates of the detected pixels (*)
MaxX	1	Bounding box right margin
MinX	1	Bounding box left margin
MaxY	1	Bounding box bottom margin
MinY	1	Bounding box top margin

(*) Values can be divided by the number pixels to obtain the center point.

Figure 3. Data Format of the Results of Region Growing Algorithm

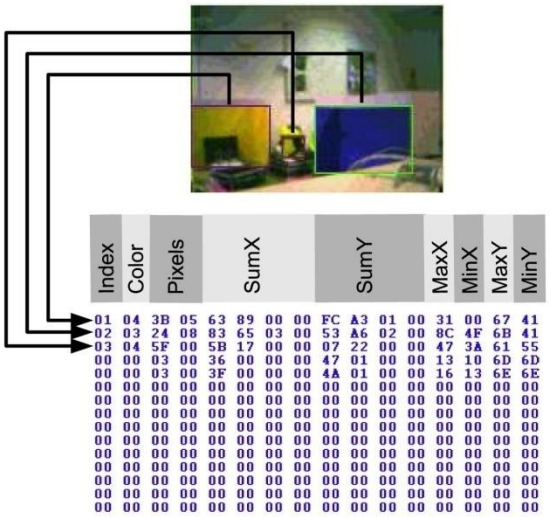
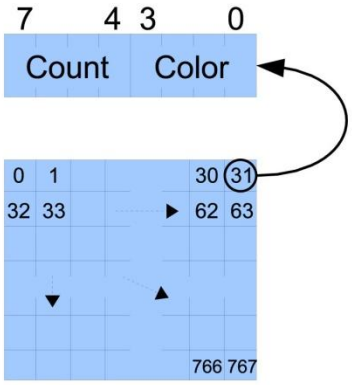


Figure 4. Results of the Region Growing Algorithm

To access the results, the instruction *READ_REG* should be used. Up to 15 regions can be read from the address range 0x10 to 0xFF using the *READ_REG* (0x02) instruction. Following example shows the produced output of the module according to a given image.

3.3.2 On-line Griding Algorithm

After invoking the griding algorithm with *CAP_GRID* instruction, it compresses the image into a grid of 32*24. Each cell is a representative for a 5x5 square on the original image. For each cell one byte is calculated which contains the flowing information. The lower 4 bits determine which color was the most prior detected in the square. The higher 4 bits are the number of pixels occupied with this color. If over 15 pixels are detected the count remains 15.



The algorithms results 768 bytes of data. These can be read using the *READ_GRID* instruction. Note that the given address is internally multiplied by 16 to increase the access range. It is therefore required that a multi-byte read operation with at least 16 bytes be used.

Following example shows the result of a the griding algorithm. Note that only the color is visualized.