

 Universidade do Porto Faculdade de Engenharia <b>FEUP</b>	<b>Faculdade de Engenharia da Universidade do Porto</b> Licenciatura em Engenharia Informática e Computação <b>Robótica – Disciplina Opcional do 4º Ano</b> <b>Exame Final – Época Normal</b>		<b>Duração:</b> <b>3h00m</b>
	<b>Docentes:</b>	<b>Eugénio da Costa Oliveira e Luís Paulo Reis</b>	<b>Data:</b>
<b>Nome:</b>			

### PLANEAMENTO EM ROBÓTICA (4 Val.)

1. São dadas as situações Inicial: em(a,c) em(b,d) e braço\_livre, e Final desejada: em(c,b). O espaço de trabalho só tem 3 posições possíveis. Os Operadores distinguem colocar em e retirar de cima de outro bloco ou mesa (4 operadores: retirar, colocar, pegar, pousar). Os estados explicitam as “torres” existentes.

- a- Descreva detalhadamente, usando um Planeador tipo-STRIPS, parte da dedução do plano até concluir que esse Planeador não seria eficiente neste caso. Explique porquê.
- b- Que método utilizaria para criar um plano eficiente. Explique porquê. Descreva um plano final (sem necessidade de o derivar do método anterior).
- c- Crie a Tabela Triangular para esse plano e explique-a com detalhe.
- d- Diga que axiomas de enquadramento usou na descrição do mundo.

### PROGRAMAÇÃO DE ROBÔS EM OPEN-R SDK (4 Val.)

Responda às seguintes questões tendo como base o exemplo anexo “Sample: Ball Tracking Head” do OPEN-R SDK.

2.1 Explique detalhadamente como é que o robô efectua o processamento e análise de imagem de forma a detectar a posição da bola e como vira o pescoço para lá, centrando a bola na imagem. Explique, apresentando as alterações que teria de executar no código, como alteraria o programa de forma a unicamente detectar uma cruz laranja em vez de detectar formas esféricas cor-de-rosa?

2.2. Suponha que deseja colocar vários robôs AIBO a circularem em caravana. O primeiro robô deve-se movimentar evitando obstáculos e evitando os restantes robôs. Os restantes robôs devem-se movimentar seguindo-se uns aos outros, i.e. o segundo segue o primeiro, o terceiro segue o segundo e assim sucessivamente. Indique detalhadamente como poderia construir rapidamente esta aplicação com o auxílio dos códigos base construídos nos Mini-Projectos da disciplina de Robótica. Apresente as alterações de código necessárias para construir esta aplicação.

### COMPETIÇÕES ROBÓTICAS (3 Val.)

3.1. Compare as ligas de simulação 2D e 3D, robôs pequenos, robôs médios, robôs com pernas e simulação de resgate e salvamento do campeonato do mundo de Futebol Robótico – RoboCup a nível do tipo de metodologias Multi-Agente a utilizar na construção de uma equipa competitiva.

3.2. Descreva sucintamente as diferenças de hardware entre um robô para participar nas competições robóticas: Condução Autónoma do Festival Nacional de Robótica (FNR),

Seguimento de Pista do FNR, Micro-Rato e Robô Bombeiro (usando todas as regras). Em que competições lhe parece possível a um robô ERS7 participar? Justifique.

## ROBÓTICA (10 Val.)

4. Responda a 10 das seguintes 12 questões em 5-10 linhas:

1. No âmbito da robótica baseada em comportamentos explique a expressão que define a função de Coordenação de um Robô.
2. Indique justificando a que cores correspondem os seguintes valores de RGB: (0,0,0), (252,250,253), (255,255,0), (0,0,120), (50,50,50), (255,128,0).
3. Suponha que deseja aplicar metodologias aprendizagem no processo de localização de um robô ERS7 num campo da liga de robôs com pernas. Para tal possui uma câmara extra colocada sobre o campo e uma aplicação que recebe a imagem dessa câmara e fornece a localização (quase exacta) do robô no campo. Esta aplicação é executada num computador externo munido de uma placa de comunicação Wireless. Explique resumidamente o processo que seguiria.
4. Caracterize um Planeador Não-Linear, Hierárquico. O que é a Regressão de um Objectivo sobre um Operador? Exemplifique.
5. Quais as três principais competições robóticas realizadas anualmente em Portugal, que provas incluem, em que data aproximada e em que local se realizaram as respectivas edições de 2004 e 2005?
6. Na prova de condução autónoma, explique resumidamente como poderia realizar a detecção da sinalização fornecida pelo semáforo. Suponha que semáforo já está centrado na imagem fornecida pela câmara do robô.
7. Indique, explicando duas opções de hardware que permitam efectuar o regresso na prova Micro-Rato, explicando resumidamente a sua ligação ao software do robô.
8. Apresente em pseudo-código um algoritmo que lhe permita concluir a versão mais simples (i.e sem extras) da prova robô-bombeiro. Suponha que o robô possui dois sensores de proximidade na frente, orientados a  $60^\circ$ , um sensor que permite detectar a vela, uma ventoinha e dois motores.
9. Indique três dos Robôs humanóides mais avançados da actualidade, resumindo as suas características
10. Resuma as características do LEGO MINDSTORMS Robotics Invention System 2.0, indicando o tipo de sensores e actuadores que disponibiliza.
11. Em que consiste o DARPA Grand-Challenge? Que sensores e actuadores pensa que são vitais para poder participar neste “challenge”?
12. Utilizando o código FCP Agent fornecido em anexo, e supondo disponíveis as funções *OpponentPos(Num)* e *TeammatePos(Num)* que retornam respectivamente as posições dos oponentes e companheiros de equipa no campo, implemente uma estratégia simples de ataque em equipa em que os robôs virtuais, imediatamente após receberem a bola, passam a bola ao companheiro mais avançado no campo (segundo o eixo dos xx). Caso não exista nenhum companheiro nestas condições então o jogador envia a bola para o canto da área contrária mais distante da sua posição actual. Caso o jogador esteja dentro da área adversária remata à baliza (aleatoriamente para um dos cantos da baliza).

## MemPortugal.C

```
#define NUM_FORM 1 //Formations (like 433Open, 422, etc)
#define NUM_PLTYPES 5 //Player Types

void PortugallInfo::Initialize()
{
    ReadFormations();
    SetGlobalFormation(0);
    for(int i=1;i<=SP_team_size;i++) FM_Positioning[i]=i;
}

/*****
/***** Formation Analysis *****/
/*****/
void PortugallInfo::ReadFormations()
{
    int i,t,k;
    FILE *fich;
    char line1[512], *str;
    if ((fich=fopen("formations.conf","r"))==NULL) {
        fprintf(stderr,"Formation Configuration File does not Exist!\n"); return; }

    for(i=0; i < NUM_FORM; i++) {
        fgets(line1, 200, fich); fgets(line1, 30, fich);
        fgets(line1, 200, fich); str = &line1[0];
        for(t=1; t<=11; t++) FM_Position[i][0][t].x = get_float(&str);
        fgets(line1, 200, fich); str = &line1[0];
        for(t=1; t<=11; t++) FM_Position[i][0][t].y = get_float(&str);
        fgets(line1, 200, fich); str = &line1[0];
        for(t=1; t<=11; t++) FM_PlayerType[i][t] = get_int(&str);
        fgets(line1, 200, fich); str = &line1[0];
        for(k=0; k<NUM_PLTYPES; k++) FM_Attraction[i][k].x = get_float(&str);
        fgets(line1, 200, fich); str = &line1[0];
        for(k=0; k<NUM_PLTYPES; k++) FM_Attraction[i][k].y = get_float(&str);
        fgets(line1, 200, fich); str = &line1[0];
        for(k=0; k<NUM_PLTYPES; k++) FM_BehindBall[i][k] = get_int(&str);
        fgets(line1, 200, fich); str = &line1[0];
        for(k=0; k<NUM_PLTYPES; k++) FM_MinX[i][k] = get_float(&str);
        fgets(line1, 200, fich); str = &line1[0];
        for(k=0; k<NUM_PLTYPES; k++) FM_MaxX[i][k] = get_float(&str);
    }
    fclose(fich);
}

// Why not use situations? What should be the tactic and formation for each situation?
// How to define different player types?
void PortugallInfo::DecideFormation(void)
{
    SetGlobalFormation(0);
}

/*****
/***** Communication *****/
/*****/
// Only players that think they have important things to say, should talk!
void PortugallInfo::DoCommunication(void) // Communicate World State
{
    if (ClockStopped || BallVelocityValid())>=0.9 && (CurrentTime.t+2) % 2 == 0 )
        send_game_status(); // Send World State
}

void PortugallInfo::say_something(char *message) // Say Message
{
    SayAction.type = CMD_say;
    SayAction.time = CurrentTime;
    sprintf(SayAction.command, "(say %s)",message);
}

// Why not communicate the players positions and velocities and useful events?
void PortugallInfo::send_game_status() // Send Game Status
{
    char str[512], str_aux[512];
    if(!MyConf()) return;

    sprintf(str_aux,"world_status %1.2f %1.2f ", MyX(),MyY());
    strcpy(str,str_aux);
    if (BallPositionValid()) {
        sprintf(str_aux,"1 %1.2f %1.2f %1.2f ",BallPositionValid(), BallX(),BallY());
        strcat(str,str_aux);
        if (BallVelocityValid())
```

```

        sprintf(str_aux,"1 %1.2f %1.2f %1.2f ",BallVelocityValid(),
                BallAbsoluteVelocity().x, BallAbsoluteVelocity().y);
    else sprintf(str_aux,"0 ");
    strcat(str,str_aux);
}
else {
    sprintf(str_aux,"0 ");
    strcat(str,str_aux);
}
say_something(str);
}

// To compress and encode this, is a very good idea!
void PortugallInfo::parse_world_status(char *message) // Parse Game Status
{
    char *str = &message[0];
    float x, y, pivotx, pivoty, conf, vx, vy, vconf;

    get_word(&str);
    pivotx = get_float(&str); pivoty = get_float(&str);
    if (get_int(&str)) { // Ball position
        conf = get_float(&str); x = get_float(&str); y = get_float(&str);
        if (get_int(&str)) { // Ball Velocity
            vconf = get_float(&str); vx = get_float(&str); vy = get_float(&str);
            HearBall(x,y,conf, vx,vy,vconf,Vector(x-pivotx,y-pivoty).mod(), CurrentTime);
        }
        else HearBall(x,y,conf, Vector(x-pivotx,y-pivoty).mod(), CurrentTime);
    }
}

// To use an online coach may be very useful!
void PortugallInfo::parse_received_message(int direction, char *message)
{
    switch(message[0]) { case 'w': parse_world_status(message); }
}

/*****
/***** Individual Decision Making *****/
/*****

// Lots of things to explore here! This is really a basic decision mechanism...
void PortugallInfo::DecideAction(void) // Individual Decision Making
{
    if(!MyConf() || !BallPositionValid()) {scan_field_with_body(); return;}
    RefreshPositioningMatrix(); // analyses interceptions
    if (BallKickable()) DoBallMove(); // I have the ball? What to do with it?
    else DoPositioning(); // Go to best Position!
}

// Pass, Forward, Dribble, Hold, Shoot to different places, [Reis and Lau 2001]...
void PortugallInfo::DoBallMove(void) // Ball Possession: Situation With Ball
{
    smart_kick_hard(AngleToFromBody(Vector(52.5,0.0)), KM_Moderate); //Shoot to the Goal!
}

// It is not a good idea to always intercept... Sometimes is better to keep the position...
// Think about other ball recovery actions like mark pass lines, mark opponents, cover goal.
void PortugallInfo::DoPositioning(void) // Ball Recovery and Positioning: Situations Without
ball
{
    if (PO_MyInterc-2<=PO_TeammateInterc && PO_MyInterc<30 || PO_MyInterc<10)
        get_ball(); // Intercept the Ball
    else DoSBSPPositioning(); // Strategic Position
}

// This is not really SBSPP because situations are not considered.
void PortugallInfo::DoSBSPPositioning(void) // Strategic Situation
{
    go_to_point(SBSPPosition(MyNumber),1.5+fabs(BallX()-MyX())/10.0,PowerConserveStamina());
}

// This is also basic. Reason about the importance of spending stamina at each moment!
float PortugallInfo::PowerConserveStamina(void) // Stamina Management
{
    if (MyInterceptionAble() &&
        (abs(PO_MyInterc-PO_OpponInterc)<3 || PO_MyInterc<4)) return 100.0; //Ball is disputed
    if(MyStamina()-150.0<SP_recover_dec_thr*SP_stamina_max)
        return 30.0*(MyRecovery()); // don't loose recovery!
    return 100.0;
}

```

```

// Only basic information included. Why not use interception matrixes
// (with values for all players) and reason using that information.
// Use distances on the reasoning! Use similar structure to Ball Possession information!
void PortugallInfo::RefreshPositioningMatrix() //Interception Analysis
{
    if (FastestOpponentToBall() != Unum_Unknown) { // Cycles to ball of fastest opponent
        PO_OpponInterc = OpponentInterceptionNumberCycles(FastestOpponentToBall());
    }
    else { PO_OpponInterc = 150; }
    if (FastestTeammateToBall() != Unum_Unknown) { // Cycles to ball of fastest teammate
        PO_TeammateInterc = TeammateInterceptionNumberCycles(FastestTeammateToBall());
    }
    else { PO_TeammateInterc = 200; }
    if (MyInterceptionAble()) { // My number of cycles to ball
        PO_MyInterc = MyInterceptionNumberCycles();
    }
    else { PO_MyInterc = 250; }
}

/*****
/***** SBSP - Situation Based Strategic Positioning *****/
/*****

// Calculates the Strategic Positioning using a very simple SBSP implementation
// Improve this using situations, attractions by points or lines in some situations, etc.
Vector PortugallInfo::SBSPPosition(int t)
{
    Vector Position = FormationGetPosition(t);
    float FactorX = FormationGetAttraction(t).x;
    float FactorY = FormationGetAttraction(t).y;
    Position.x = Position.x + BallX() * FactorX;
    Position.y = Position.y + BallY() * FactorY;
    if (fabs(Position.y) > 24.0) Position.y = Position.y - 0.5*FactorY*BallY();
    Position = PositionToOnsidePosition(Position);
    if (FormationGetBehindBall(t) && BallX() < Position.x) Position.x = BallX();
    if (Position.x > FormationGetMaxX(t)) Position.x = FormationGetMaxX(t);
    if (Position.x < FormationGetMinX(t)) Position.x = FormationGetMinX(t);
    return Position;
}

/*****
/***** Normal Player Central Module *****/
/*****

// This is a very simple player. Use different things like DPRE - Dynamic Positioning and Role
// Exchange that enables the players to switch their positionings in a given formation or
// SLM - Strategic Looking that enables the players to look at the best spot in the field
void PortugallInfo::NormalPlayer()
{
    static int players_moved = FALSE;
    if (PlayMode==PM_Before_Kick_Off || PlayMode==PM_Half_Time || PlayMode==PM_Extended_Time)
        if (players_moved) return;
        else {
            players_moved = TRUE;
            if (FormationGetPosition(MyNumber).x < 0.0) move(FormationGetPosition(MyNumber));
            else move(-1.0,FormationGetPosition(MyNumber).y);
            return;
        }
    players_moved = FALSE;

    DecideFormation(); // Formation Analysis
    DoCommunication(); // Communicate
    DecideAction(); // Individual Decision Making
}

```

## Formations.Conf

```

0 // Formation 0 *****/
-45.0 -13.0 -14.0 -14.0 -13.0 -5.0 0.0 0.0 10.0 15.0 10.0 //posx
0.0 16.0 5.0 -5.0 -16.0 0.5 10.0 -10.0 -0.5 20.0 -20.0 //posy
0 2 1 1 2 2 2 2 3 3 // PT_GOA PT_DEF PT_MID... //Type
0.1 0.5 0.5 0.5 0.5 //AttractionX
0.1 0.25 0.25 0.25 0.25 //AttractionY
1 1 0 0 0 //BehindBall
-52.5 -45.0 -45.0 -30.0 -45.0 //MinX
-30.0 0.0 35.0 42.0 42.0 //MaxX
1 // Formation 1 *****/
-45.0 -11.0 -12.0 -12.0 -11.0 3.0 -4.0 3.0 15.0 15.0 //posx
0.0 17.0 6.0 -6.0 -17.0 25.0 10.0 -10.0 -25.0 8.0 -10.0 //posy
0 2 1 1 2 2 2 2 3 3 // PT_GOA PT_DEF PT_MID... //Type
0.1 0.5 0.5 0.5 0.5 //AttractionX
0.1 0.25 0.25 0.25 0.25 //AttractionY
1 1 0 0 0 //BehindBall
-52.5 -45.0 -45.0 -30.0 -45.0 //MinX
-30.0 0.0 35.0 42.0 42.0 //MaxX

```

## BallTrackingHead.h

```
#include <OPENR/OObject.h>
#include <OPENR/OSubject.h>
#include <OPENR/OObserver.h>
#include <OPENR/ODataFormats.h>
#include <OPENR/OFbkImage.h>
#include <BallTrackingHeadData.h>
#include "def.h"

enum BallTrackingHeadState { BTHS_IDLE, BTHS_START, BTHS_HEAD_ZERO_POS,
    BTHS_LEGS_SLEEPING, BTHS_SEARCHING_BALL, BTHS_TRACKING_BALL};

static const char* const FBK_LOCATOR = "PRM:/r1/c1/c2/c3/i1-FbkImageSensor:F1";
static const char* const JOINT_LOCATOR[] = {
    "PRM:/r1/c1-Joint2:j1", // HEAD TILT
    "PRM:/r1/c1/c2-Joint2:j2" // HEAD PAN
};

class BallTrackingHead : public OObject {
public:
    BallTrackingHead();
    virtual ~BallTrackingHead() {}

    OSubject* subject[numOfSubject];
    OObserver* observer[numOfObserver];

    virtual OStatus DoInit (const OSystemEvent& event);
    virtual OStatus DoStart (const OSystemEvent& event);
    virtual OStatus DoStop (const OSystemEvent& event);
    virtual OStatus DoDestroy(const OSystemEvent& event);

    void NotifyImage(const ONotifyEvent& event);
    void NotifySensor(const ONotifyEvent& event);
    void NotifyMovingHeadResult(const ONotifyEvent& event);
    void NotifyMovingLegsResult(const ONotifyEvent& event);
    void NotifyLostFoundSoundResult(const ONotifyEvent& event);

private:
    static const size_t NUM_COMMAND_VECTOR = 4;
    static const size_t NUM_SENSOR_VECTOR = 2;
    static const size_t NUM_JOINTS = 2;
    static const size_t NUM_FRAMES = 5;
    static const int TILT_INDEX = 0;
    static const int PAN_INDEX = 1;
    static const byte BALL_THRESHOLD = 10;
    static const int FOUND_THRESHOLD = 2;
    static const int LOST_THRESHOLD = 5;
    static const OCdtChannel BALL_CDT_CHAN = ocdtCHANNEL0;
    static const double FIELD_VIEW_H = 57.6;
    static const double FIELD_VIEW_V = 47.8;

    void OpenPrimitives();
    void NewCommandVectorData();
    void PlaySound(BallTrackingHeadCommandType type);

    void SetCdtVectorDataOfPinkBall();
    bool CentroidAndDeltaAngle(const OFbkImage& cdtImage,
        int* xcentroid, int* ycentroid, double* delta_pan, double* delta_tilt);

    void InitSensorIndex(OSensorFrameVectorData* sensorVec);
    void GetPanTiltAngle(longword frameNum, double* pan, double* tilt);

    void SearchBall();
    void TrackBall(longword frameNum, double delta_pan, double delta_tilt);
    void MoveHead(double s_pan, double s_tilt, double e_pan, double e_tilt,
        double *r_pan, double *r_tilt, double pan_limit, double tilt_limit);
    void SetJointValue(RCRegion* rgn, int idx, double start, double end);
    RCRegion* FindFreeRegion();

    BallTrackingHeadState ballTrackingHeadState;
    OPrimitiveID fbkID;
    OPrimitiveID jointID[NUM_JOINTS];
    RCRegion* region[NUM_COMMAND_VECTOR];
    bool initSensorIndex;
    int sensoridx[NUM_JOINTS];
    list<RCRegion*> sensorRegions;
    double lastRefPan;
    double lastRefTilt;
};
```

## BallTrackingHead.cc

```
#include <OPENR/OPENRAPI.h>
#include <OPENR/OSyslog.h>
#include <OPENR/core_macro.h>
#include <BallTrackingHeadData.h>
#include "BallTrackingHead.h"

BallTrackingHead::BallTrackingHead() : ballTrackingHeadState(BTHS_IDLE),
                                       fbkID(oprimitiveID_UNDEF), initSensorIndex(false),
                                       sensorRegions(), lastRefPan(0), lastRefTilt(0)
{
    for (int i = 0; i < NUM_JOINTS; i++) jointID[i] = oprimitiveID_UNDEF;
    for (int i = 0; i < NUM_COMMAND_VECTOR; i++) region[i] = 0;
    for (int i = 0; i < NUM_JOINTS; i++) sensoridx[i] = -1;
}

OStatus BallTrackingHead::DoInit(const OSystemEvent& event)
{
    OSYSDEBUG(("BallTrackingHead::DoInit()\n"));
    NEW_ALL_SUBJECT_AND_OBSERVER;
    REGISTER_ALL_ENTRY;
    SET_ALL_READY_AND_NOTIFY_ENTRY;

    OpenPrimitives();
    NewCommandVectorData();
    SetCdtVectorDataOfPinkBall();
    return oSUCCESS;
}

OStatus BallTrackingHead::DoStart(const OSystemEvent& event)
{
    OSYSDEBUG(("BallTrackingHead::DoStart()\n"));
    BallTrackingHeadCommand headcmd(BTHCMD_MOVE_TO_ZERO_POS);
    subject[sbjMovingHead]->SetData(&headcmd, sizeof(headcmd));
    subject[sbjMovingHead]->NotifyObservers();

    BallTrackingHeadCommand legscmd(BTHCMD_MOVE_TO_SLEEPING);
    subject[sbjMovingLegs]->SetData(&legscmd, sizeof(legscmd));
    subject[sbjMovingLegs]->NotifyObservers();

    ballTrackingHeadState = BTHS_START;
    ENABLE_ALL_SUBJECT;
    ASSERT_READY_TO_ALL_OBSERVER;
    return oSUCCESS;
}

OStatus BallTrackingHead::DoStop(const OSystemEvent& event)
{
    OSYSDEBUG(("BallTrackingHead::DoStop()\n"));
    ballTrackingHeadState = BTHS_IDLE;
    DISABLE_ALL_SUBJECT;
    DEASSERT_READY_TO_ALL_OBSERVER;
    return oSUCCESS;
}

OStatus BallTrackingHead::DoDestroy(const OSystemEvent& event)
{
    DELETE_ALL_SUBJECT_AND_OBSERVER;
    return oSUCCESS;
}

void BallTrackingHead::NotifyMovingHeadResult(const ONotifyEvent& event)
{
    OSYSDEBUG(("BallTrackingHead::NotifyMovingHeadResult()\n"));
    if (ballTrackingHeadState == BTHS_IDLE) return; // do nothing

    BallTrackingHeadResult* result = (BallTrackingHeadResult*)event.Data(0);
    if (result->status == BTH_SUCCESS) {
        if (ballTrackingHeadState == BTHS_START) {
            ballTrackingHeadState = BTHS_HEAD_ZERO_POS;
        } else if (ballTrackingHeadState == BTHS_LEGS_SLEEPING) {
            SearchBall();
            ballTrackingHeadState = BTHS_SEARCHING_BALL;
        }
    } else {
        OSYSLOG1((osyslogERROR, "%s : %s %d", "BallTrackingHead::NotifyMovingHeadResult()",
            "FAILED. result->status", result->status));
    }
    observer[event.ObsIndex()->AssertReady();
}

```

```

void BallTrackingHead::NotifyMovingLegsResult(const ONotifyEvent& event)
{
    OSYSDEBUG(("BallTrackingHead::NotifyMovingLegsResult()\n"));
    if (ballTrackingHeadState == BTHS_IDLE) return; // do nothing
    BallTrackingHeadResult* result = (BallTrackingHeadResult*)event.Data(0);
    if (result->status == BTH_SUCCESS) {
        if (ballTrackingHeadState == BTHS_START) {
            ballTrackingHeadState = BTHS_LEGS_SLEEPING;
        } else if (ballTrackingHeadState == BTHS_HEAD_ZERO_POS) {
            SearchBall();
            ballTrackingHeadState = BTHS_SEARCHING_BALL;
        }
    } else {
        OSYSLOG1((osyslogERROR, "%s : %s %d", "BallTrackingHead::NotifyMovingLegsResult()",
            "FAILED. result->status", result->status));
    }
    observer[event.ObsIndex()->AssertReady();
}

void BallTrackingHead::NotifyLostFoundSoundResult(const ONotifyEvent& event)
{
    OSYSDEBUG(("BallTrackingHead::NotifyLostFoundSoundResult()\n"));
    if (ballTrackingHeadState == BTHS_IDLE) return; // do nothing
    BallTrackingHeadResult* result = (BallTrackingHeadResult*)event.Data(0);
    if (result->status != BTH_SUCCESS) {
        OSYSLOG1((osyslogERROR, "%s : %s %d", "BallTrackingHead::NotifyLostFoundSoundResult()",
            "FAILED. result->status", result->status));
    }
    observer[event.ObsIndex()->AssertReady();
}

void BallTrackingHead::OpenPrimitives()
{
    OStatus result;
    for (int i = 0; i < NUM_JOINTS; i++) {
        result = OPENR::OpenPrimitive(JOINT_LOCATOR[i], &jointID[i]);
        if (result != oSUCCESS) {
            OSYSLOG1((osyslogERROR, "%s : %s %d", "BallTrackingHead::OpenPrimitives()",
                "OPENR::OpenPrimitive() FAILED", result));
        }
    }
    result = OPENR::OpenPrimitive(FBK_LOCATOR, &fbkID);
    if (result != oSUCCESS) {
        OSYSLOG1((osyslogERROR, "%s : %s %d", "BallTrackingHead::OpenPrimitives()",
            "OPENR::OpenPrimitive() FAILED", result));
    }
}

void BallTrackingHead::NewCommandVectorData()
{
    OStatus result;
    MemoryRegionID    cmdVecDataID;
    OCommandVectorData* cmdVecData;
    OCommandInfo*    info;

    for (int i = 0; i < NUM_COMMAND_VECTOR; i++) {
        result = OPENR::NewCommandVectorData(NUM_JOINTS, &cmdVecDataID, &cmdVecData);
        if (result != oSUCCESS) {
            OSYSLOG1((osyslogERROR, "%s : %s %d", "BallTrackingHead::NewCommandVectorData()",
                "OPENR::NewCommandVectorData() FAILED", result));
        }

        region[i] = new RCRegion(cmdVecData->vectorInfo.memRegionID,
            cmdVecData->vectorInfo.offset, (void*)cmdVecData, cmdVecData->vectorInfo.totalSize);
        cmdVecData->SetNumData(NUM_JOINTS);

        for (int j = 0; j < NUM_JOINTS; j++) {
            info = cmdVecData->GetInfo(j);
            info->Set(odataJOINT_COMMAND2, jointID[j], NUM_FRAMES);
        }
    }
}

void BallTrackingHead::PlaySound(BallTrackingHeadCommandType type)
{
    OSYSDEBUG(("BallTrackingHead::PlaySound()\n"));
    BallTrackingHeadCommand cmd(type);
    subject[subjLostFoundSound]->SetData(&cmd, sizeof(cmd));
    subject[subjLostFoundSound]->NotifyObservers();
}

```

## BallTrackingHead\_Image.cc

```
#include <OPENR/OPENRAPI.h>
#include <OPENR/OSyslog.h>
#include "BallTrackingHead.h"

void BallTrackingHead::NotifyImage(const ONotifyEvent& event)
{
    static int found = 0;
    static int lost = 0;

    int xc, yc; // centroid
    double d_pan, d_tilt; // delta angle

    if (ballTrackingHeadState == BTHS_IDLE) return; // do nothing

    OFbkImageVectorData* imageVec = (OFbkImageVectorData*)event.Data(0);

    OFbkImageInfo* info = imageVec->GetInfo(ofbkimageLAYER_C);
    byte* data = imageVec->GetData(ofbkimageLAYER_C);
    OFbkImage cdtImage(info, data, ofbkimageBAND_CDT);

    if (ballTrackingHeadState == BTHS_SEARCHING_BALL) {

        if (cdtImage.ColorFrequency(BALL_CDT_CHAN) >= BALL_THRESHOLD) {
            found++;
        } else {
            found = 0;
            SearchBall();
        }

        if (found == FOUND_THRESHOLD) {
            OSYSPRINT(("### BALL FOUND ### \n"));
            PlaySound(BTHCMD_PLAY_FOUND_SOUND);
            found = 0;
            CentroidAndDeltaAngle(cdtImage, &xc, &yc, &d_pan, &d_tilt);
            TrackBall(info->frameNumber, d_pan, d_tilt);
        }

    } else if (ballTrackingHeadState == BTHS_TRACKING_BALL) {

        if (cdtImage.ColorFrequency(BALL_CDT_CHAN) >= BALL_THRESHOLD) {
            lost = 0;
            CentroidAndDeltaAngle(cdtImage, &xc, &yc, &d_pan, &d_tilt);
            TrackBall(info->frameNumber, d_pan, d_tilt);
        } else {
            lost++;
        }

        if (lost == LOST_THRESHOLD) {
            OSYSPRINT(("### BALL LOST ### \n"));
            PlaySound(BTHCMD_PLAY_LOST_SOUND);
            SearchBall();
            lost = 0;
        }

    }

    observer[event.ObsIndex()->AssertReady();
}

void BallTrackingHead::SetCdtVectorDataOfPinkBall()
{
    OSYSDEBUG(("BallTrackingHead::SetCdtVectorDataOfPinkBall()\n"));

    OStatus result;
    MemoryRegionID cdtVecID;
    OCdtVectorData* cdtVec;
    OCdtInfo* cdt;

    result = OPENR::NewCdtVectorData(&cdtVecID, &cdtVec);
    if (result != oSUCCESS) {
        OSYSLOG1((osyslogERROR, "%s : %s %d", "ImageObserver::SetCdtVectorDataOfPinkBall()",
            "OPENR::NewCdtVectorData() FAILED", result));
        return;
    }

    cdtVec->SetNumData(1);

    cdt = cdtVec->GetInfo(0);
    cdt->Init(fbkID, BALL_CDT_CHAN);
}
```

```

// cdt->Set(Y_segment, Cr_max, Cr_min, Cb_max, Cb_min)
cdt->Set( 0, 230, 150, 190, 120);
cdt->Set( 1, 230, 150, 190, 120);
cdt->Set( 2, 230, 150, 190, 120);
cdt->Set( 3, 230, 150, 190, 120);
cdt->Set( 4, 230, 150, 190, 120);
cdt->Set( 5, 230, 150, 190, 120);
cdt->Set( 6, 230, 150, 190, 120);
cdt->Set( 7, 230, 150, 190, 120);
cdt->Set( 8, 230, 150, 190, 120);
cdt->Set( 9, 230, 150, 190, 120);
cdt->Set(10, 230, 150, 190, 120);
cdt->Set(11, 230, 150, 190, 120);
cdt->Set(12, 230, 150, 190, 120);
cdt->Set(13, 230, 150, 190, 120);
cdt->Set(14, 230, 150, 190, 120);
cdt->Set(15, 230, 150, 190, 120);
cdt->Set(16, 230, 150, 190, 120);
cdt->Set(17, 230, 150, 190, 120);
cdt->Set(18, 230, 150, 190, 120);
cdt->Set(19, 230, 150, 190, 120);
cdt->Set(20, 230, 160, 190, 120);
cdt->Set(21, 230, 160, 190, 120);
cdt->Set(22, 230, 160, 190, 120);
cdt->Set(23, 230, 160, 190, 120);
cdt->Set(24, 230, 160, 190, 120);
cdt->Set(25, 230, 160, 190, 120);
cdt->Set(26, 230, 160, 190, 120);
cdt->Set(27, 230, 160, 190, 120);
cdt->Set(28, 230, 160, 190, 120);
cdt->Set(29, 230, 160, 190, 120);
cdt->Set(30, 230, 160, 190, 120);
cdt->Set(31, 230, 160, 190, 120);

result = OPENR::SetCdtVectorData(cdtVecID);
if (result != oSUCCESS) {
    OSYSLOG1((osyslogERROR, "%s : %s %d", "ImageObserver::SetCdtVectorDataOfPinkBall()",
        "OPENR::SetCdtVectorData() FAILED", result));
}

result = OPENR::DeleteCdtVectorData(cdtVecID);
if (result != oSUCCESS) {
    OSYSLOG1((osyslogERROR, "%s : %s %d", "ImageObserver::SetCdtVectorDataOfPinkBall()",
        "OPENR::DeleteCdtVectorData() FAILED", result));
}
}

bool BallTrackingHead::CentroidAndDeltaAngle(const OFbkImage& cdtImage,
    int* xcentroid, int* ycentroid, double* delta_pan, double* delta_tilt)
{
    int w      = cdtImage.Width();
    int h      = cdtImage.Height();
    byte* ptr  = cdtImage.Pointer();

    int xsum   = 0;
    int ysum   = 0;
    int npixels = 0;

    // Last line is not used because of TagInfo.
    for (int y = 0; y < h-1; y++) {
        for (int x = 0; x < w; x++) {
            if (*ptr++ & BALL_CDT_CHAN) {
                xsum += x; ysum += y; npixels++;
            }
        }
    }

    if (npixels != 0) {
        *xcentroid = xsum / npixels;
        *ycentroid = ysum / npixels;
        *delta_pan  = -1.0 * FIELD_VIEW_H * (*xcentroid - w/2.0) / w;
        *delta_tilt = -1.0 * FIELD_VIEW_V * (*ycentroid - h/2.0) / h;
        return true;
    } else {
        *xcentroid = 0; *ycentroid = 0; *delta_pan = 0.0; *delta_tilt = 0.0;
        return false;
    }
}
}

```

## BallTrackingHead\_Joint.cc

```
#include <OPENR/OPENRAPI.h>
#include <OPENR/OUnits.h>
#include <OPENR/OSyslog.h>
#include <OPENR/ODEbug.h>
#include "BallTrackingHead.h"

void BallTrackingHead::SearchBall()
{
    static double phase      = 0.0;
    const double deltaPhase = 0.03;
    const double amplitude  = 70.0; // degrees

    OSYSDEBUG(("BallTrackingHead::SearchBall()\n"));
    if (ballTrackingHeadState != BTHS_SEARCHING_BALL) {
        ballTrackingHeadState = BTHS_SEARCHING_BALL;
        if (lastRefPan < 0.0) { phase = lastRefPan / amplitude; }
        else { phase = M_PI - lastRefPan / amplitude; }
    }

    const double panDeltaLimit = deltaPhase * amplitude; // degrees / 32ms
    const double tiltDeltaLimit = 1.0; // degrees / 32ms
    double pan = amplitude * sin(phase);
    double tilt = 0.0;
    phase += deltaPhase;
    MoveHead(lastRefPan, lastRefTilt, pan, tilt, &lastRefPan, &lastRefTilt,
             panDeltaLimit, tiltDeltaLimit );
}

void BallTrackingHead::TrackBall(longword frameNum, double delta_pan, double delta_tilt)
{
    if (ballTrackingHeadState==BTHS_SEARCHING_BALL){ballTrackingHeadState=BTHS_TRACKING_BALL;}
    const double panDeltaLimit = 5.0; // degrees / 32ms
    const double tiltDeltaLimit = 5.0; // degrees / 32ms
    double pan, tilt;
    GetPanTiltAngle(frameNum, &pan, &tilt);
    MoveHead(lastRefPan, lastRefTilt, pan + delta_pan, tilt + delta_tilt,
             &lastRefPan, &lastRefTilt, panDeltaLimit, tiltDeltaLimit );
}

void BallTrackingHead::MoveHead(double s_pan, double s_tilt, double e_pan, double e_tilt,
                                double *r_pan, double *r_tilt, double pan_limit, double tilt_limit)
{
    RCRegion* rgn = FindFreeRegion();

    if (e_pan > s_pan + pan_limit) { e_pan = s_pan + pan_limit; }
    else if (e_pan < s_pan - pan_limit) { e_pan = s_pan - pan_limit; }
    *r_pan = e_pan;
    if (e_tilt > s_tilt + tilt_limit) { e_tilt = s_tilt + tilt_limit; }
    else if (e_tilt < s_tilt - tilt_limit) { e_tilt = s_tilt - tilt_limit; }
    *r_tilt = e_tilt;

    SetJointValue(rgn, PAN_INDEX, s_pan, e_pan);
    SetJointValue(rgn, TILT_INDEX, s_tilt, e_tilt);
    subject[sbjJoint]->SetData(rgn);
    subject[sbjJoint]->NotifyObservers();
}

void BallTrackingHead::SetJointValue(RCRegion* rgn, int idx, double start, double end)
{
    OCommandVectorData* cmdVecData = (OCommandVectorData*)rgn->Base();
    OCommandInfo* info = cmdVecData->GetInfo(idx);
    info->Set(odataJOINT_COMMAND2, jointID[idx], NUM_FRAMES);
    OCommandData* data = cmdVecData->GetData(idx);
    OJointCommandValue2* jval = (OJointCommandValue2*)data->value;

    double delta = end - start;
    for (int i = 0; i < NUM_FRAMES; i++) {
        double dval = start + (delta * i) / (double)NUM_FRAMES;
        jval[i].value = oradians(dval);
    }
}

RCRegion* BallTrackingHead::FindFreeRegion()
{
    for (int i = 0; i < NUM_COMMAND_VECTOR; i++) {
        if (region[i]->NumberOfReference() == 1) return region[i];
    }
    return 0;
}

```

## BallTrackingHead\_Sensor.cc

```
#include <OPENR/OSyslog.h>
#include "BallTrackingHead.h"

void BallTrackingHead::NotifySensor(const ONotifyEvent& event)
{
    RCRegion* rgn = event.RCData(0);

    if (initSensorIndex == false) {
        OSensorFrameVectorData* sv = (OSensorFrameVectorData*)rgn->Base();
        InitSensorIndex(sv);
        initSensorIndex = true;
    }
    if (sensorRegions.size() == NUM_SENSOR_VECTOR) {
        sensorRegions.front()->RemoveReference();
        sensorRegions.pop_front();
    }
    rgn->AddReference();
    sensorRegions.push_back(rgn);
    observer[event.ObsIndex()->AssertReady();
}

void BallTrackingHead::InitSensorIndex(OSensorFrameVectorData* sensorVec)
{
    // jointID[] is already initialized in BallTrackingHead::OpenPrimitives().
    for (int i = 0; i < NUM_JOINTS; i++) {
        for (int j = 0; j < sensorVec->vectorInfo.numData; j++) {
            OSensorFrameInfo* info = sensorVec->GetInfo(j);
            if (info->primitiveID == jointID[i]) {
                sensoridx[i] = j;
                OSYSDEBUG("[%2d] %s\n", sensoridx[i], JOINT_LOCATOR[i]);
                break;
            }
        }
    }
}

void BallTrackingHead::GetPanTiltAngle(longword frameNum, double* pan, double* tilt)
{
    list<RCRegion*>::iterator iter = sensorRegions.begin();
    list<RCRegion*>::iterator last = sensorRegions.end();
    OSensorFrameVectorData* sv;
    OSensorFrameInfo* info;
    OSensorFrameData* data;

    while (iter != last) {
        sv = (OSensorFrameVectorData*)(*iter)->Base();
        info = sv->GetInfo(sensoridx[PAN_INDEX]);
        longword fn = info->frameNumber;
        for (int i = 0; i < info->numFrames; i++) {
            if (fn == frameNum) {
                data = sv->GetData(sensoridx[PAN_INDEX]);
                *pan = degrees((double)data->frame[i].value / 1000000.0);
                data = sv->GetData(sensoridx[TILT_INDEX]);
                *tilt = degrees((double)data->frame[i].value / 1000000.0);
                return;
            }
            fn++;
            if (fn > oframeMAX_NUMBER) fn = 1;
        }
        ++iter;
    }

    // If sensor data at frameNum is not found, return latest sensor data.
    RCRegion* rgn = sensorRegions.back();
    sv = (OSensorFrameVectorData*)rgn->Base();
    info = sv->GetInfo(sensoridx[PAN_INDEX]);
    data = sv->GetData(sensoridx[PAN_INDEX]);
    *pan = degrees((double)data->frame[info->numFrames-1].value / 1000000.0);
    data = sv->GetData(sensoridx[TILT_INDEX]);
    *tilt = degrees((double)data->frame[info->numFrames-1].value / 1000000.0);
}
```