```
/*
 * Subcribes to laser data and camera data,
 * asks user for desired distance and real size of "blob".
 * Follows "blob" with obstacle avoidance.
 *
 */

#include <libplayerc++/playerc++.h>
#include <iostream>
#include "args.h"

int main(int argc, char **argv)
{
   // Interpret command line args into gHostname, gPort, other global options
   parse_args(argc,argv);

   // we throw exceptions on creation if we fail
   try
   {
      // Use player namespace to get easy access to player functions
      using namespace PlayerCc;
      // Use standard namespace to get easy access to cout, etc
      using namespace std;

      // Connect to player server on the robot
      PlayerClient robot(gHostname, gPort);
      // Subsribe to position proxy on robot server
      Position2dProxy pp(&robot, gIndex);
      // Subscribe to laser proxy on robot server
      LaserProxy lp(&robot, gIndex);

      Graphics2dProxy  gp(&robot, gIndex);
      //Color(uint8_t red,  uint8_t green,  uint8_t blue,  uint8_t alpha);
      gp.Color(0, 255,  200,  0);

      BlobfinderProxy bp(&robot, gIndex);

      cout << robot << endl;

      cout<< std::max(3,5) << endl;

      // Set player server message behavior
      robot.SetReplaceRule(true,-1,-1,-1);
      robot.SetDataMode(gDataMode);

      robot.Read();
```

```
int count = lp.GetCount();

// wait for real data
while( count == 0 )
{
   robot.Read();
   count = lp.GetCount();
}

double maxRange=lp.GetMaxRange();
double angRes = lp.GetScanRes();

pp.SetMotorEnable (true);
//pp.ResetOdometry();

robot.Read();

double xPos = pp.GetXPos();
double yPos = pp.GetYPos();
double theta = pp.GetYaw();

double R = 0.25;

double newSpeed = 0;
double newTurnRate = 0;

// Speed limits
double maxSpeed = 0.3; // in m/s
double maxTurnRate = 0.7; // in rad/s

// goal position
double xGoal = xPos;
double yGoal = yPos;
double closeEnough = 0.05;

//field of view
double fov=dtor(60);

//resolution of camera
double resCam=bp.GetWidth();

double realSize=0;

double desDistance=0;
double desiredHeading=0;
double distToGoal=0;
```

```
double distToGoal1=0;
double desiredHeading1=0;

double angleSafe=0;
int trouble=-1;
double angleTrouble=2*M_PI;
double distTrouble=0;
double buffer=.3;


cout<<"Desired distance from object to robot= ";
cin>>desDistance;

cout<<"Real size of colored object= ";
cin>>realSize;

robot.Read();

// go into read-think-act loop
for(;;)
{
   // ------- Read ------------
   // this blocks until new data comes; 10Hz by default
   robot.Read();
   robot.Read();
   robot.Read();
   //while( robot.data_requested == 1 )
   {
      robot.Read();
   }

   // number of sensor measurements from laser
   count = lp.GetCount();

   // resolution of laser
   angRes = lp.GetScanRes();

   resCam=bp.GetWidth();

   cout<<bp;

   int midWidth=0;
   int oy=0;
   int dy=0;
   //int ox=0;
   //int dx=0;
```

```
for(unsigned int i=0;i<bp.GetCount();i++)
{
   midWidth=bp.GetBlob(i).x;
   oy=bp.GetBlob(i).top;
   dy=bp.GetBlob(i).bottom;
   //ox=bp.GetBlob(i).left;
   //dx=bp.GetBlob(i).right;
}

// position of robot
xPos = pp.GetXPos();
yPos = pp.GetYPos();
theta = normalize(pp.GetYaw());

double distanceToBlob=0;
double angleDifference=0;


if (gDoObstacleAvoid && oy==0 && dy==0)
{
   distanceToBlob = sqrt(pow(xGoal-xPos,2) + pow(yGoal-yPos,2));
   angleDifference = normalize(atan2(yGoal-yPos,xGoal-xPos) - theta);

   cout<<"no cam data xGoal= "<<xGoal<<endl;
   cout<<"no cam data yGoal= "<<yGoal<<endl;
   cout<<"xPos:" <<xPos<<endl;
   cout<<"yPos:" <<yPos<<endl;

   if( distToGoal1 < closeEnough )
   {
      cout<<"No data"<<endl;
      pp.SetSpeed(0,.2);
      continue;

   }
}

// -------- Think ----------

else
{

   desiredHeading=0;
   distToGoal=0;
   distToGoal1=0;
```

```
        desiredHeading1=0;

        double distanceDifference=-desDistance+((realSize*resCam/2)/(dy-
oy)/tan(fov/2));
        //cout<<"distanceDifference: "<<distanceDifference<<endl;

        distanceToBlob = ((realSize*resCam/2)/(dy-oy)/tan(fov/2))-R;

        //change of distance
        if (abs(distanceDifference)<=.1)
        {
           distToGoal1=0;
        }
        else if (abs(distanceDifference)>=.1)
        {
           distToGoal1=distanceDifference;
        }

        angleDifference=(resCam/2-midWidth)*fov/resCam;
        cout<< "resCam " << resCam << "; midWidth " << midWidth << "; fov " <<
fov << endl;
        //cout<<"angleDifference: "<<angleDifference<<endl;

        //change of angle

        yGoal=(distanceDifference+desDistance)*sin(theta+angleDifference)+yPos;
        xGoal=(distanceDifference+desDistance)*cos(theta+angleDifference)+xPos;

        /*DrawPolygon(player_point_2d_t pts[],
              int count,
              bool filled,
              player_color_t fill_color);

        player_point_2d_t points[1];
        points[0].px = xGoal;
        points[0].py = yGoal;
        //gp.Clear();
        gp.DrawPoints(points, 1);
        */


        cout<<"theta= "<<theta<<endl;
        cout<<"xPos= "<<xPos<<endl;
        cout<<"yPos= "<<yPos<<endl;
        cout<<"xGoal= "<<xGoal<<endl;
        cout<<"yGoal= "<<yGoal<<endl;
```

```
}


   desiredHeading1=angleDifference;
   bool SafePathToGoal=true;
   double angCutoff=atan2(R+buffer,(distanceToBlob));

   angleSafe=0;
   trouble=-1;
   angleTrouble=2*M_PI;
   distTrouble=maxRange;

   cout<<"angleDifference= "<<angleDifference<<endl;
   cout<<"distanceToBlob= "<<distanceToBlob<<endl;


// loop through laser distances
for( int j = 0; j < count; j++ )
{

   double angleJ=angRes*(j-count/2);
   double drectJ=0;

   int angleType;

   /*if (fabs(angleJ)<angCutoff)
   {
      drectJ=fabs((distanceToBlob)/cos(angleJ));
      angleType=1;
   }
   */
   //else if (fabs(angleJ) < M_PI/2.0)

   if (fabs(angleJ)>angCutoff && fabs(angleJ) < M_PI/2.0)
   {
      drectJ=fabs((R+buffer)/sin(angleJ));
      angleType=2;

      if (lp[j]<distTrouble && lp[j] < drectJ)
      {
         SafePathToGoal=false;
         distTrouble=lp[j];
         angleTrouble=angleJ;
         trouble=j;
      }
```

```
    }

}

if (SafePathToGoal || !gDoObstacleAvoid)
{
    cout<<"Safe Path"<<endl;
    desiredHeading=desiredHeading1;
    distToGoal=distToGoal1;
}
else
{
    cout<<"distTrouble: "<<distTrouble<<endl;
    cout<<"angleTrouble: "<<angleTrouble<<endl;

    /*
    player_point_2d_t points[1];
    //points[0].px = xGoal;
    //points[0].py = yGoal;
    points[0].px = distTrouble*cos(angleTrouble+theta)+xPos;
    points[0].py = distTrouble*sin(angleTrouble+theta)+yPos;
    //gp.Clear();
    gp.DrawPoints(points, 1);
    */

    if (distTrouble<(R+buffer))
    {
        angleSafe=(M_PI/2)-atan(distTrouble/(R+buffer));


    }
    else
    {
        angleSafe=asin((R+buffer)/distTrouble);
        cout<<"angleSafe: "<<angleSafe<<endl;
    }

    if (angleTrouble <= 0)
    {
        desiredHeading = angleDifference+angleSafe;
        distToGoal=distToGoal1;
    }
    else
    {
        desiredHeading = angleDifference-angleSafe;
        distToGoal=distToGoal1;
```

```
            }
            cout<<"angleSafe: "<<angleSafe<<endl;
            cout<<"desiredHeading: "<<desiredHeading<<endl;
            cout<<"distToGoal1: "<<distToGoal1<<endl;

            if( !(desiredHeading < M_PI && desiredHeading > -M_PI) )
            {
               exit(0);
            }
         }


         // calculate speed and turn rate
         newSpeed = maxSpeed*limit(2*(distToGoal),-1.0,1.0);
         //newSpeed *= limit(minObsDist,0.0,1.0);
         newSpeed *= limit((0.6 - fabs(desiredHeading))/0.6,0.0,1.0);
         newTurnRate = desiredHeading;

         // Limit speeds based on defined maximums
         newSpeed = limit( newSpeed, -maxSpeed/2, maxSpeed );
         newTurnRate = limit(newTurnRate,-maxTurnRate,maxTurnRate);

         cout << "speed: " << newSpeed << " m/s "
              << "turn: " << newTurnRate << " rad/s"
              << endl << endl;



         // ---------- Act ------------
         // write commands to robot
         pp.SetSpeed(newSpeed, newTurnRate);


      }
   }
   catch (PlayerCc::PlayerError e)
   {
      std::cerr << e << std::endl;
      return -1;
   }
}
```