Aria Homework 1

Noah Harvey

November 10, 2014

Code

asn1.cpp

```
/*
 * tst.cpp is part of mecharia.
 *
 * mecharia is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * mecharia is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with mecharia.  If not, see <http://www.gnu.org/licenses/>.
 */

/**
 * @file tst.cpp
 *
 * Aria test/learning file
 *
 * @author Dr. Matthew Marshall (mmarshall@spsu.edu)
 * @author Noah Harvey (nharvey@spsu.edu)
 *
 * @copyright GNU Public License 2
 */
#include <Aria.h>
#include "ariaTypedefs.h"
#include "ariaUtil.h"
#include "ArAction.h"

const int SQRDIST = 2000; /**< square shapre distance is 2 meters */

const double ps[4] = {0,SQRDIST,SQRDIST,0};
typedef struct
{
        unsigned char xcnt:2;
        unsigned char ycnt:2;
```

```
}cntr_t;

class ArActionGotoCust : public ArActionGotoStraight
{
        public:
                ArActionGotoCust() : ArActionGotoStraight("goto",10000){};
                virtual ~ArActionGotoCust() {};
                virtual ArActionDesired* fire(ArActionDesired);
};

/** @brief custom fire action to have robot rotote before it moves linearly
 *
 * code copied from ArActionGotoStraight.h
 */
ArActionDesired *ArActionGotoCust::fire(ArActionDesired currentDesired)
{
  double angle;
  double dist;
  double distToGo;
  double vel;

  // if we're there we don't do anything
  if (myState == STATE_ACHIEVED_GOAL || myState == STATE_NO_GOAL)
    return NULL;

  ArPose goal;
  if (!myUseEncoderGoal)
  {
    goal = myGoal;
    myDistTravelled += myRobot->getPose().findDistanceTo(myLastPose);
    myLastPose = myRobot->getPose();
  }
  else
  {
    goal = myRobot->getEncoderTransform().doTransform(myEncoderGoal);
    myDistTravelled += myRobot->getEncoderPose().findDistanceTo(myLastPose);
    myLastPose = myRobot->getEncoderPose();
  }

  if (myJustDist)
  {
    distToGo = myDist - myDistTravelled;
    dist = fabs(distToGo);
  }
  else
  {
    dist = myRobot->getPose().findDistanceTo(goal);
  }

  if (((myJustDist && distToGo <= 0) ||
       (!myJustDist && dist < myCloseDist))
```

```
      && ArMath::fabs(myRobot->getVel() < 5))
  {
    if (myPrinting)
      ArLog::log(ArLog::Normal, "Achieved goal");
    myState = STATE_ACHIEVED_GOAL;
    myDesired.setVel(0);
    myDesired.setDeltaHeading(0);
    return &myDesired;
  }

  // see where we want to point
  angle = myRobot->getPose().findAngleTo(goal);
  if (myBacking)
    angle = ArMath::subAngle(angle, 180);

  myDesired.setHeading(angle);

        /**
         * added by Noah Harvey
         * rotate towards our target before we start moving
         */
        if(myRobot->getRotVel() == 0)
        {
                // if we're close, stop
          if ((myJustDist && distToGo <= 0) ||
              (!myJustDist && dist < myCloseDist))
          {
            myDesired.setVel(0);
            vel = 0;
          }
          else
          {
            vel = sqrt(dist * 200 * 2);
            if (vel > mySpeed)
              vel = mySpeed;
            if (myBacking)
              vel *= -1;
            myDesired.setVel(vel);
          }
        }
        else
                myDesired.setVel(0);

  if (myPrinting)
    ArLog::log(ArLog::Normal, "dist %.0f angle %.0f vel %.0f",
               dist, angle, vel);

  return &myDesired;
}

void asn1(ArRobot* robot)
```

```
{
        char i;
        cntr_t cntr;
        cntr.xcnt = 1;
        cntr.ycnt = 0;

        if(!robot)
                return;

        ArLog::log(ArLog::Normal,"Starting Assigment 1");

        /** set up robot actions */
        ArActionGotoCust gotopnt;
        gotopnt.setCloseDist(0);
        gotopnt.setRobot(robot); //uneeded
        robot->addAction(&gotopnt,100);

        /** turn on the motors */
        robot->enableMotors();

        /** set the target points in turn */
        for(i = 0; i < 4; i++)
        {
                gotopnt.setGoal(ArPose(ps[cntr.xcnt++],ps[cntr.ycnt++]),false,
                    false);
                while(!gotopnt.haveAchievedGoal());
        }

        robot->disableMotors();
        ArLog::log(ArLog::Normal,"End of Assigment 1");

        return;
}
```