```
#CS4610: Robotic Science and Systems
-----------------------------------

##Overview
----------
This lab involved programming the robot to pick up a "flag" using its arm.

##Compiling
----------
run make

##Running
---------
for pickup code:
  ./run-class Arm x-coordinate-of-flag y-coordinate-of-flag
for key control:
  ./run-class Control

##Problems
----------
When gripper length longer than 260 mm, after pressing ???a??? or ???d??? you
can see the gripper extend itself obviously to maintain the y coordination in
the previous robot frame. Otherwise the gripper barely move after passing
???a??? or ???d??.

The code for deciding if a point is reachable is a bit buggy.  It can generally
prevent the arm from reaching bad points, but it has a lot of false negatives.


##Who Did What?
---------------
Nick was absent for the first lab for an interview.  As such, Qimeng and Zach
started the project.  Zach started by writing the inverse kinematics code and
a lot of the starting code in Arm.  Qimeng implemented the forward kinematics
and the key control code.  Nick implemented the point interpolation and
cleaned up the code in both files.
```

Demo grading (partly qualitative): B manual (keyboard) I/K broken for Y motion

Overall grade: 94/100

```
/*******************************************************************************
*
* Robotic Systems and Science: CS4610
* Lab 3: Arm Kinematics and Grasping
* Zach Ferland, Nick Jones, Qimeng Song
*
*******************************************************************************/

package l3;

import ohmm.*;
import static ohmm.OHMM.*;
import ohmm.OHMM.AnalogChannel;
import static ohmm.OHMM.AnalogChannel.*;
import ohmm.OHMM.DigitalPin;
import static ohmm.OHMM.DigitalPin.*;

import java.math.*;
import java.lang.Math;
import java.util.Arrays;
import java.io.*;
import java.util.ArrayList;

/**
* Arm provides a method of controlling the Ohmm arm.
* @author Zach Ferland, Nick Jones
* @version 3/12/2014
*/
public class Arm {

  //length from joint 0 to joint 1 (mm)
  public static final float len0 = 99;

  //lenght from joint 1 to to joint 2 (wrist) (mm)
  public static final float len1 = 74;

  /**
  * Inverse kinematics
  * @param gx - desired x coordinate of gripper in robot frame
  * @param gz - desired z coordinate of gripper in robot frame
  * @return {Sholder, Elbow, Wrist} angles in that order in the array
  */
  public static float[] invK(float gx, float gz) {

    //A1,A2, A3)
    float[] angles = new float[3];

    //differences in frames of each orgin frame of joint
    float wz = 19f;
    float wx = 74f;

    float sz = 91f;
    float sx = 38f;

    float rz = gz - wz - sz;
    float rx = gx - wx - sx;

    //reach side + calculate this after
    float r = (float)Math.sqrt((rz * rz) + (rx * rx));

    float c1 = ((r * r) - (len0 * len0) - (len1 * len1)) / (2 * len0 * len1);

    float s1 = (float) Math.sqrt(Math.abs(1 - (c1 * c1)));

    //some float to double conversion issues
    float a1 = (float)Math.atan2(s1,c1);

    float alpha = (float)Math.atan2((-1 * rz), rx);

    //negative 2 for kink down
    float cb = ((len1 * len1) - (len0 * len0) - (r * r)) / (-2 * len0 * r);
    float sb = (float) Math.sqrt(Math.abs(1 - (cb * cb)));

    float beta = (float)Math.atan2(sb,cb);

    float a0 = alpha - beta;

    float a2 = 0 - a0 - a1;
```

In code like this where you just have a few floating point variables it is usually no problem to use double (64bit) for everything instead of float (32bit). You get more precision and that can actually matter in a surprising number of cases.

Only when you are storing a lot (millions or more) of numbers does it usually make sense to consider float instead of double. This happens commonly when doing 3D graphics code for example.

```
  angles[0] = a0;
  angles[1] = a1;
  angles[2] = a2;


  return angles;
}



/**
* Set the Arm angles
* @param angles - The angles to set
*                   angles[0] - sholder
*                   angles[1] - elbow
*                   angles[2] - wrist
* @param ohmm - The robot to control
*/
public static void setArm(float[] angles, OHMMDrive ohmm) {
  ohmm.armSetAllJointsRad(angles[0], angles[1], angles[2]);
  while(ohmm.armActive());
}



/**
* Drive to a point in the world coordinate frame
* @param x - x coordinate of point in world coordinate frame
* @param y - y coordinate of point in world coordinate frame
* @param ohmm - The robot to control
*/
public static void driveToPoint(float x, float y, OHMMDrive ohmm) {
  Arm.turnToPoint(x, y, ohmm);
  float dist = (float) Math.sqrt((x * x) + (y * y));
  float actual = dist - 250f;
  ohmm.driveStraight(actual);
  while(ohmm.driveGetQueue() != 0);
  //driveGoal
  //calcualte distance drive goal, with vertex.
  //ohmm df dist
}



/**
* Turn to face a point in the world coordinate frame
* @param x - x coordinate of point in world coordinate frame
* @param y - y coordinate of point in world coordinate frame
* @param ohmm - The robot to control
*/
public static void turnToPoint(float x, float y, OHMMDrive ohmm) {
  //starts at origin so diff is x y of goal
  float angle = (float) Math.atan2(y, x);
  ohmm.driveTurn(angle);
  while(ohmm.driveGetQueue() != 0);
}



/**
* Drive the robot back to the origin (0,0) in world frame
* @param x - current robot x coordinate in world frame
* @param y - current robot y coordinate in world frame
* @param ohmm - robot to control
*/
public static void returnToOrigin(float x, float y, OHMMDrive ohmm) {
  ohmm.driveTurn(3.1415926f);
  while(ohmm.driveGetQueue() != 0);
  float dist = (float) Math.sqrt((x * x) + (y * y));
  ohmm.driveStraight(dist);
  while(ohmm.driveGetQueue() != 0);
}



/**
* 3D distance formula
```

```
* @param x1 - point 1 x
* @param y1 - point 1 y
* @param z1 - point 1 z
* @param x2 - point 2 x
* @param y2 - point 2 y
* @param z2 - point 2 z
* @return distance from point 1 to point 2
*/
public static Double distance(Double x1, Double y1, Double z1,
                              Double x2, Double y2, Double z2) {
  return Math.sqrt(Math.pow(x1 - x2, 2) +
                   Math.pow(y1 - y2, 2) +
                   Math.pow(z1 - z2, 2));
}

/**
* Move the gripper to the point given in a straight line.
* @param x - x coordinate to move to in robot frame
* @param y - y coordinate to move to in robot frame
* @param z - z coordinate to move to in robot frame
* @param ohmm - The robot to control
* @return true iff the robot was able to move to the point
*/
public static Boolean moveGripToPoint(float x, float y, float z,
  OHMMDrive ohmm) {

  if (isReachable(x, y, z)) {
    float[] gripPos = Control.currentGripperLoc(ohmm);
    ArrayList<Point3D> interp =
      interpolateLine(new float[]{gripPos[0], 0f, gripPos[1]},
                      new float[]{x, 0f, z}, 3.0);

    for(Point3D p : interp)
      System.out.println(p);

    for(Point3D p : interp) {
      float[] ik  = Arm.invK(p.x, p.z);
      //if (isReachableAngle(ik[0], ik[1], ik[2]))
        Arm.setArm(ik, ohmm);
      //else
      //  System.out.println("Point not reachable");
      while(ohmm.armActive());
      float[] pos = Control.currentGripperLoc(ohmm);
      System.out.printf("point:\n\tx: %f z: %f\n", p.x, p.z);
      System.out.printf("actual:\n\tx: %f z: %f\n\n", pos[0], pos[1]);
    }

    return true;
  }
  else {
    return false;
  }
}

/**
* Find a series of points in between the current gripper position (x, z) and
* the desired position.  Frame doesnt matter, but it will stay the same from
* input to return.
*
* @param current - current gripper x, y, z positions in mm
* @param target  -  target gripper x, y, z positions in mm
* @return A sequence of points that will take the gripper from start to target
*         in small steps.  List is ordered with next position at index 0 and
*         target position at last index.
*/
public static ArrayList<Point3D> interpolateLine(float[] current,
  float[] target, Double stepDist) {
  ArrayList<Point3D> result = new ArrayList<Point3D>();

  Double currentX = new Double(current[0]);
  Double currentY = new Double(current[1]);
  Double currentZ = new Double(current[2]);
  Double targetX  = new Double(target[0]);
  Double targetY  = new Double(target[1]);
  Double targetZ  = new Double(target[2]);
  Double dx = targetX - currentX;
  Double dy = 0.0;//targetY - currentY;
  Double dz = targetZ - currentZ;
```

```
  Double startDist = distance(currentX, currentY, currentZ, targetX, targetY,
    targetZ);

  Double xStep;
  Double yStep;
  Double zStep;
  if (startDist != 0.0) {
    xStep = stepDist * dx / startDist;
    yStep = stepDist * dy / startDist;
    zStep = stepDist * dz / startDist;
  }
  else
    return result;

  System.out.printf("step: %f %f", xStep, zStep);

  // Add all intermediate points
  while (distance(currentX, currentY, currentZ, targetX, targetY, targetZ)
          > stepDist)
    result.add(new Point3D((currentX += xStep).floatValue(),
                           (currentY += yStep).floatValue(),
                           (currentZ += zStep).floatValue()));

  // Add target point
  result.add(new Point3D(target[0], target[1], target[2]));

  return result;
}



/**
* Open the robot arm gripper
* @param ohmm - The robot to control
*/
public static void openGrip(OHMMDrive ohmm) {
  ohmm.armSetGripper(1f);
  while(ohmm.armActive());
}


/**
* Close the robot arm gripper
* @param ohmm - The robot to control
*/
public static void closeGrip(OHMMDrive ohmm) {
  ohmm.armSetGripper(0f);
  while(ohmm.armActive());
}



/**
* Is the 3D point in robot frame reachable?
* @param x - x coordinate
* @param y - y coordinate
* @param z - z coordinate
* @return true iff the point is reachable by the gripper
*/
public static Boolean isReachable(float x, float y, float z) {
  return z >= 10 &&
         z < 320 &&
         x > 150 &&
         Math.sqrt(x * x + y * y + z * z) < 300;
}

/**
* Are the angles of the arms reachable?
* @param The sholder angle in radians
* @param The elbow angle in radians
* @param The wrist angle in radians
* @return true iff the angles are within bounds.
*/
public static Boolean isReachableAngle(float a1, float a2, float a3) {
  return -Math.PI / 2.0 < a1 && a1 < Math.PI / 4.0 &&
        -Math.PI * 5.0 / 9.0 < a2 && a2 < Math.PI * 5.0 / 6.0 &&
        -Math.PI / 2.0 < a3 && a3 < Math.PI / 2.0;
}
```

This is not a correct way to determine reachability. The I/K algorithm we developed in lecture includes a few key steps where you can check reachability.

(-3)

```
  /**
  * ENTRY POINT
  */
  public static void main(String[] argv) throws IOException {

    //create the OHMM object
    // OHMMDrive ohmm = (OHMMDrive) OHMM.makeOHMM(argv);
    OHMMDrive ohmm =
      (OHMMDrive) OHMM.makeOHMM(new String[]{"-r", "/dev/ttyACM1"});

    if (ohmm == null) System.exit(0);

    // Set up robot
    ohmm.armEnable(true);
    ohmm.driveSetPose(0f, 0f, 0f);
    ohmm.armHome();
    while(ohmm.armActive());

    // Parse (x, y) position of target
    float x;
    float y;
    try {
      x = Float.parseFloat(argv[0]);
      y = Float.parseFloat(argv[1]);
    }
    catch (Exception e) {
      System.err.println("Usage: ./run-class Arm x-coordinate y-coordinate");
      ohmm.close();
      System.exit(0);
      return;
    }

    // Run Sequence
    Arm.driveToPoint(x, y, ohmm);
    Arm.openGrip(ohmm);
    moveGripToPoint(Control.currentGripperLoc(ohmm)[1], 0, 20f, ohmm);
    moveGripToPoint(250f, 0, 20f, ohmm);
    Arm.closeGrip(ohmm);
    ohmm.armHome();
    while (ohmm.armActive());
    float[] pose = ohmm.driveGetPose();
    Arm.returnToOrigin(pose[0], pose[1], ohmm);
    ohmm.driveTurn(-pose[2]);

    ohmm.close();
  }
}
```

```
/*******************************************************************************
 *
 * Robotic Systems and Science: CS4610
 * Lab 3: Arm Kinematics and Grasping
 * Zach Ferland, Nick Jones, Qimeng Song
 *
 *******************************************************************************/

package l3;

import java.io.IOException;

import ohmm.*;

/**
 * Control provides a way of manipulating the Ohmm arm using the keyboard
 * @author Qimeng Song
 * @version 1/12/2014
 *
 * Controls:
 *  w - move the gripper +d mm in the world frame x direction
 *  s - move the gripper -d mm in the world frame x direction
 *  a - move the gripper +d mm in the world frame y direction
 *  d - move the gripper -d mm in the world frame y direction
 *  r - move the gripper +d mm in the world frame z direction
 *  f - move the gripper -d mm in the world frame z direction
 *  q - set d as min(d + 5, 20)
 *  z - set d as max(d - 5, 5)
 *
 * Assume that robot has initial world frame pose of (0, 0, 0)
 */
public class Control {

  /**
   * Read key input from user modifying gripper location based upon it
   * @param ohmm - The robot to control
   */
  private static void readKey(OHMMDrive ohmm) throws IOException,
  InterruptedException{
    float gx = 0;
    float gz = 0;
    float[] gR = currentGripperLoc(ohmm);
    gx = gR[0];  //gripper x coordination to the Robot frame
    gz = gR[1];  //gripper z coordination to the Robot frame
    double alpha = 0;
    //    double theta = 0;
    float d = 5f;
    for (;;) {
      System.out.print("hit any key: ");
      int c;

      c = ConsoleNonblocking.getChar();
      System.out.println("int value "+c+"; ASCII char \'"+((char)c)+"\'");


      switch (c) {
      case ConsoleNonblocking.ESC :
        System.exit(0);
      case 'w' :
        gx += d;
        if(Arm.isReachable(gx, 0, gz)){
          Arm.setArm(Arm.invK(gx, gz), ohmm);
        }
        else{
          gx -= d;
          System.out.println("not reachable");
        }
        break;
      case 's' :
        gx -= d;
        if(Arm.isReachable(gx, 0, gz)){
          Arm.setArm(Arm.invK(gx, gz), ohmm);
        }
        else{
          gx += d;
          System.out.println("not reachable");
        }
        break;
```

```
    case 'a' :
      alpha = Math.atan2(d, gx);
      gx += d * Math.sin(alpha);
      Arm.setArm(Arm.invK(gx, gz), ohmm);
      while (ohmm.armActive()){};
      ohmm.driveTurn((float) alpha);
      break;
    case 'd' :
      alpha = Math.atan2(d, gx);
      gx += d * Math.sin(alpha);
      Arm.setArm(Arm.invK(gx, gz), ohmm);
      while (ohmm.armActive()){};
      ohmm.driveTurn((float) -alpha);
      break;
    case 'r' :
      gz += d;
      Arm.setArm(Arm.invK(gx, gz), ohmm);
      break;
    case 'f' :
      gz -= d;
      Arm.setArm(Arm.invK(gx, gz), ohmm);
      break;
    case 'q' :
      d = Math.min(d + 5, 20f);
      System.out.println(d);
      break;
    case 'z' :
      d = Math.max(d - 5, 5);
      System.out.println(d);
      break;
    }
    while (ohmm.armActive()){};
    System.out.println("gx: "+gx+", gz: "+gz +", alpha: "+alpha);
  }

}



/**
 * Get the current gripper location in the Robot coordinate frame
 * @param ohmm - The robot to control
 * @return The x, y coordinates of the gripper tip in the Robot frame
 */
public static float[] currentGripperLoc(OHMMDrive ohmm){
  float[] gR = new float[2];
  float wz = 19f;
  float wx = 74f;
  float sz = 91f;
  float sx = 38f;
  float l0 = Arm.len0;
  float l1 = Arm.len1;

  float[] theta = ohmm.armGetAllJointsRad();
  float c0 = (float)Math.cos(theta[0]);
  float s0 = (float)Math.sin(theta[0]);

  float c012 = (float)Math.cos(theta[0]+theta[1]+theta[2]);
  float s012 = (float)Math.sin(theta[0]+theta[1]+theta[2]);

  float c01 = (float)Math.cos(theta[0]+theta[1]);
  float s01 = (float)Math.sin(theta[0]+theta[1]);

  gR[0] = c012*wx +s012*wz+c01*l1+c0*l0+sx;
  gR[1] = -s012*wx+c012*wz-s01*l1-s0*l0+sz;
  return gR;
}


/**
 * ENTRY POINT
 */
public static void main(String[] argv) throws IOException,
InterruptedException {
  OHMMDrive ohmm =
      (OHMMDrive) OHMM.makeOHMM(new String[]{"-r", "/dev/ttyACM1"});

  if (ohmm == null) System.exit(0);
```

```
    ohmm.armEnable(true);
    //ohmm.armSetAllJointsRad(Arm.invK(220f, 45f));
    //while (ohmm.armActive()); // Wait for arm to stop moving
    ohmm.armHome();
    while(ohmm.armActive());

    readKey(ohmm);
  }
}
```

```
/*******************************************************************************
*
* Robotic Systems and Science: CS4610
* Lab 3: Arm Kinematics and Grasping
* Zach Ferland, Nick Jones, Qimeng Song
*
*******************************************************************************/

package l3;

/**
* Point3D represents a 3D point represented by floats x y and z
* @author Nick Jones
* @version 3/12/2014
*/
public class Point3D {
  public float x; // x coordinate
  public float y; // y coordinate
  public float z; // z coordinate

    /**
    * Constructor
    * @param x - x coordinate
    * @param y - y coordinate
    * @param z - z coordinate
    */
  public Point3D(float x, float y, float z) {
    this.x = x;
    this.y = y;
    this.z = z;
  }

    /**
    * Returns a string representation of this 3d point
    * @return "x: xcoord y: ycoord z: zcoord\n"
    */
    public String toString() {
        return "x: " + x + " y: " + y + " z: " + z + "\n";
    }
}
```