CSCI–4190 Introduction to Robotic Algorithms, Spring 2002
Lab 1: out April 5, to be completed by April 15

Robot programming lab

For this lab, you will learn the basic procedures for using the robot and will program it to perform a simple task. This lab will be "supervised," meaning that a TA will be in the lab to get you started, answer any questions, and to verify that you have completed the lab.

Here are the steps you must take to complete this lab:

- Sign up for a one hour time slot with your lab partner.
- *Before* coming to the lab, read this handout completely and write a draft of your program for the basic lab exercise.

  Because your time for this lab will be limited, it is important that you come to the lab prepared.
- Come to the lab (Amos Eaton 106) at your appointed time. The TA will check your draft program (see above), review basic procedures for using the robot and get you started on the lab exercise. At the end of the hour, you will demonstrate your program.

You will get full credit for the lab for attending and completing the basic lab exercise. There are some extensions to the basic exercise for which there is "extra credit."

Lab exercises

The basic lab exercise is to write a program for the robot that performs the following task:

> The robot will start pointed towards a rectangular box (unspecified size) in the middle of the room. The direction of the robot motion will be (as close as possible to) perpendicular to the side of the box.
> The robot should move forward until it is 0.5 meters from the box, turn right, travel around the box, maintaining an offset of 0.5 meters, and stop at the same position where it started its circuit around the box.

It should be possible to complete this basic task using only straight-line motion of the robot. (In other words, you shouldn't have to use feedback from the sensors to adjust the linear motion of the robot. You may find that you need to adjust the angle of the turns to maintain the 0.5 meter offset.) You may find it useful to use both the SONAR and IR sensors.

After you have demonstrated a program that completes the basic exercise, you can attempt either or both of the following extensions for "extra credit."

Extension 1: Make the offset distance a parameter to your program.

Extension 2: Adapt your program to work when the first side of the box is not perpendicular to the robot.

An introduction to the Magellan Pro robot

The Magellan Pro is a commercial robot made by the iRobot corporation. The robot is equipped with an on-board PC running Red Hat Linux. It is 40.6cm in diameter and 25.4cm tall and has 16 SONAR sensors, 16 IR sensors, and 16 bump switches. Initial communication and diagnostics are performed using a tuning knob and LCD screen on the robot itself. It

can be controlled using a joystick as well as programs written with its own unique *Mobility Robot Integration Software*.

The Mobility software is a distributed, object-oriented toolkit for building control software for single and multi-robot systems. Mobility was written using the Common Object Request Broker Architecture (CORBA) 2.X standard Interface Definition Language (IDL). The scope of a Mobility system is contained within objects that are accessible from a top-level NamingService. The naming service can run on the robot computer or a lab computer. The naming server contains a directory of robot objects or shared support objects. A base server running on the robot directly communicates with programs through the naming service.

Programming and Running the Robot

Once you have a written and compiled your program, to run it on the robot you must do the following:

1. Turn the robot on.
2. Start the base server. (You will know it is running when you hear the sonar sensors making clicking sounds.)

   ```
   $ robot start
   ```

3. Run your compiled program from `maximal.robotics.cs.rpi.edu` using the command line

   ```
   $./myProgram -robot MagellanPro
   ```

4. When your program has finished running, stop the base server:

   ```
   $ robot stop
   ```

Robot Programming Interface

The Mobility Robot Object Model connects to different objects within the robot structure to give programmers a very general beginning. A lot of the complications have been removed for you through the implementation of a simple Robot object. The following section describes the methods available to you through this Robot object.

- `Robot(char *robotName)` - constructor for the robot class. The robot name from the command line should be used here.
- `void forward(float dist, float v)` - drives a given distance forward at the given speed.
- `void turn(float theta)` - turns the given angle in radians. Counterclockwise turns are positive. Clockwise turns are negative.
- `void setVelocity(float v, float w)` - sets forward and angular velocities. The angular velocity, w, should be entered in radians per second. Entering zero for both these parameters will stop the robot.
- `void setMaxVelocity(float v)` - the maximum allowable velocity that will be implemented in the trapezoidal velocity profile.
- `float getMaxVelocity()` - returns the velocity for the trapezoidal velocity profile.

- `void setAcceleration(float a)` - sets the acceleration that will be implemented in the trapezoidal velocity profile.
- `float getAcceleration()` - returns the acceleration for the trapezoidal velocity profile.
- `void getOdometry(float &x,float &y,float &theta)` - retrieves the location and angle(in radians) of the robot.
- `MobilityGeometry::SegmentData_var getSonar()` - returns a pointer to the mobility object containing sonar data. Sonar data is returned as a set of 16 line segments.
- `MobilityGeometry::SegmentData_var getIR()` - returns a pointer to the mobility object containing IR data. Like the sonar data, IR data is returned as a set of 16 line segments.

A Sample Program

This is a simple program that moves the robot forward until the sonar sensors detect that it is within 0.5 m from a wall.

The robot class that implements your simple robot interface is defined in robot.h. Include this in all of your programs for the lab.

```
#include <math.h>
#include "robot.h"
```

Because the Mobility structure has the potential to interface with more than one robot at a time you must specify which robot you are talking to. The robot name, Magellan Pro, should be given on the command line with the `-robot` flag.

```
int main (int argc, char *argv[])
{
  char *robotName = mbyUtility::get_option(argc,argv,"-robot");
  if (robotName == NULL)    {
    fprintf(stderr,"Need a robot name to use.\n");
    return -1;
  }

  Robot myTurtle(robotName);
```

The program sets up the variables for determining distance from the wall and a storage point for sonar data.

```
  // Now, here is a loop that continually checks robot sensors and
  // sends drive commands
  MobilityGeometry::SegmentData_var sonarData;
  unsigned long index1;    // Counts through sensor readings.
  float haltdist = 0.5;    // 50cm halts
  float tempdist;          // Computed temp dist value
  float total=0.0;         // Used for averaging sonar readings

  fprintf(stderr,
          "********** Mobility Sonar Example **********\n");
```

The main program loop. Set the robot to go, check the sensors, stop if appropriate. The program will stop under two conditions: The robot is within 0.5m from a wall or the user has requested a stop by hitting the enter key.

```
  while(1){
    // set the robot to go forward at .1m/s
    myTurtle.setVelocity(0.1, 0.0);

    // grab the sonar data
    sonarData=myTurtle.getSonar();

    for (index1 = 0; index1 < sonarData->org.length(); index1++) {
      // the three indices at the front of the robot
```

The sonar sensors are indexed 0 through 15 with 0 being the at the center on the front. The indices follow around the circumference counterclockwise.

```
      if((index1==0)||(index1==1)||(index1==15)){
        tempdist = sqrt(
                        (sonarData->org[index1].x - sonarData->end[index1].x)*
                        (sonarData->org[index1].x - sonarData->end[index1].x)+
                        (sonarData->org[index1].y - sonarData->end[index1].y)*
                        (sonarData->org[index1].y - sonarData->end[index1].y));
        total += tempdist;
      }
    }

    fprintf(stderr,"avg: %f\n\n",total/3.0);

    // the average distance from the front of the robot is less than desired
    if (total/3.0 < haltdist){
      myTurtle.setVelocity(0.0,0.0);
      return 0;
    }
    // reset the total distance
    total=0.0;

    // if the user pressed return
    if (mbyUtility::chars_ready() > 0)  {
      myTurtle.setVelocity(0.0,0.0);
      fprintf(stderr, "stopped by user\n");
      return 0;
    }
    else // Wait a small time before the next loop. 0.1 second, keep going.
        omni_thread::sleep(0,100000000);
  }
  return 0;
}
```