Lab2: Smart Luggage: Basic Perception and Feedback Control

1 Introduction

Now let’s see what we can do with the robot’s “eyes”. The robot has a scanning laser rangefinder that returns 360 range readings all around the robot. The technique used for this unique device is actually laser triangulation. Each group of 360 range readings will be called a *range scan* or range image. If you have ever wanted your luggage to just follow you at the airport, then this lab is for you.

2 Motivation

While wheel motion sensing gives the robot a sense of its own state of motion, it takes some kind of radiation (like light or sound) to get a sense of what is out there without actually touching it.

It is the capacity to react to the environment that endows robots with one form of intelligence (predicting the future is another kind). As this lab will show, even simple algorithms can endow the robot with more apparent intelligence that you might guess.

3 Exercises

3.1 Warm Up Exercise 1: Read and Display Lidar Data

Write the function or code snippet that reads the lidar data and displays it as (x,y) points repeatedly in a plot window. The lidar spins physically at 5 Hz so there is no need to update the display more than a 5 Hz. Its OK to simply call plot over and over in a loop.

To get used to MATLAB vectorized expression idiom, understand what these 3 separate bits of code are doing.

```
n = 1:6

theta = (0:9)*(pi/180)

cos(theta)
```

You can use these idioms to make your loops more efficient. It actually does not matter this time. You might want to initially plot the raw range data (while moving objects in front of the robot) versus the index into the range array, to get a sense for where the first range pixel is pointing in the robot frame and which direction of rotation corresponds to increasing index. Be deliberate and record your observations to avoid wasting your time. It takes less time to write it down than it does to fire up the robot and do it over. Once the data layout is clear, write the MATLAB function `[x y b] = irToXy(i,r).` Here, `i` is the index in the range array and `r` is the range value at that index. **Don’t forget the angular offset mentioned in class.**

This function should return the position `(x,y,th)` and bearing `th`, relative to the origin of the sensor, of the point at the end of a range pixel range `r,` occurring at index `i,` in the range array returned by the robot. To keep things simplest, define the sensor

forward axis to be parallel to the robot forward axis. The direction of the first pixel is a different matter. *Bearing* means angle relative to the sensor forward direction. Make this an actual function because you will reuse it a lot.

I created a `rangeImage` class and made this a static method but you can just make a function if you like. To do that, you put the function in its own file.

```
function [ x y th] = irToXy( i, r )
%irToXy finds position and bearing of a range pixel endpoint
%   Finds the position and bearing of the endpoint of a range pixel in
the plane.
    %   Fill in code here
end
```

Let's all use the same coordinates. It will make our lives easier. The tiny camera is the front of the robot. The robot frame has axes fixed to it. `x` is pointing forward and `y` to the left. The *bearing* to an object is an angle between $-\pi$ and π where zero bearing means along the robot x axis. Likewise robot *heading* θ with respect to the room is positive counterclockwise when looking down on the robot.

When `irToXy` is implemented, plot an "x" marker on screen at the end of every nonzero range pixel whose value is less than 1.0 meters. Plot the data on a persistent figure window (one that stays up iteration after iteration of range scan reads) in metric units. Convince yourself that the kinematics and the display are doing the right thing for each quadrant before you go any further.

3.2 Warm Up Exercise 2: Plot The Position of The Nearest Object

Write a function or code snippet that reads the lidar data and finds the position of **the nearest object**. This is defined as the position of the lidar point that is **all of the following**:

- Not smaller than 0.06 meters (to reject internal reflections from housing)
- Not larger than `maxObjectRange` (1.0 meters)
- For which `abs(bearing)< maxBearing` ($\pi/2$)

Display it as a letter 'x' plotted at position (x,y) in a persistent plot window. Move objects in front of the real robot (when it is stationary, on your desk, with nothing else in view) and convince yourself this is working correctly before you go any further.

For this and the rest of the lab, you can assume the lidar sensor origin is at the center of the robot. That is not true for most robots and many don't have a well-defined center. We will, however, account for the fact that the range scan **is rotated relative to the sensor** housing as mentioned above and in class.

3.3 Warm Up Exercise 3: Distance Servo

Write a function or code snippet that reads the lidar data and **follows the nearest object** defined above while moving **only in a straight line**. The algorithm is:

- When there is nothing in view, command no motion.
- When the object is at `idealObjectRange`,(= 0.5 meter) command no motion

- When the object is farther away, command a velocity that is proportional `(* gain)` to the excess of `objectRange` over `idealRange`.
- When the object is closer than `idealObjectRange,` move backwards with a velocity that is proportional `(* gain)` to the deficit of `objectRange` with respect to `idealRange.`
- When trying to keep the object at `idealObjectRange,` make the robot move as reasonably quickly as you can but not so quickly that it creates other problems. You can adjust the speed of response by simply adjusting the proportional gain `(gain)`.

3.4 Challenge Task: Smart Luggage

Now, improve your distance servo to make the robot **also** steer toward and maintain this distance to the nearest object, **as defined above**, even if is slightly to the left or right but within the angular range being searched. Because of noise, it will likely be necessary to assume the object is still there for a short time (say, ¾ of a second) if it momentarily disappears from the range image. However, the tradeoff is that the robot will not stop for this period in the case it has locked onto the wrong (stationary) object that, for example, reflects lidar poorly and shows up randomly.

You must search through an angle range that is 180 degrees wide (+/ 90) in front of the robot and **not** behind it. When the robot detects that the closest object is too close, it must back up.

4 Graded Demonstration and Report

Only this part is graded in the lab. Points are included in square braces [thus].

Show your robot executing the challenge task with a real time display[1] of an x for (only) the closest object in a window as the robot moves. You need to draw only the position of the object relative to the robot (i.e. the robot is invisible and at the center of the graph). Orient the graph so that up on the screen is forward in the robot frame, and use a fixed scale of +/- 2 meters on both axes. Implement this solution as a servo in curvature-speed space that converts from curvature-speed to wheel speeds using velocity kinematics. Determine the position of the tracked object, and drive toward it using curvature and speed commands in order to try to always face the object and maintain the object at the desired range (0.5 meters). We will test it using our own legs or a piece of wood about 4 inches wide as the “object”.

[1] All graphs produced for labs in this course should have (1) a title and (2) axis labels that include (3) a description of the units. Plot different signals on the same graph using (4) different colors with (5) no point markers (triangles, squares, etc.). Use a (6) legend to describe which is which. Failure to comply with these rules will result in a 20% reduction in your grade. Whenever possible, produce graphs in real time as the robot moves, but its OK to plot results at the end of the test if real-time graphics would otherwise affect performance.

[10] Robot moves continuously for single object in view.

[10] Able to keep up at 0.15 m/s.

[10] Robot goes backward when necessary.

[10] Robot turns when necessary

[10] Graph updates correctly in real time.

4.1 Report

~~Bring your report to the lab period and hand it in **before** the lab period starts. In it, include the content and answers to the questions provided below (and in other grey boxes above, if any):~~

~~[5] Describe your design for computing curvature from bearing and range to the object.~~

~~[5] You could improve your object chasing algorithms to make them respond only to moving objects. Suggest how to detect if an object is moving with respect to the room when the robot itself is moving with respect to the room.~~

~~[5] How could you improve your algorithm to make it more immune to a second person momentarily entering the field of view inside the capture radius of `maxObjectRange` (and thereby stealing your luggage!).~~

~~[10 **bonus**] It is not difficult to have the algorithm switch to driving backwards for an object behind the robot. What are the key changes that do that?~~

5 Notes and Hints

1. Most of the hints from last week still apply.
2. Note that you cannot use the debugger in MATLAB when executing a single code block. I place code blocks temporarily in their own file in when I want to test them in the debugger.
3. The simulator requires you to create walls in order to have something to “look” at. It may not be worth your time to set all that up for this lab. This is one case when its simpler to turn on the real robot and read data.
4. It helps to put the robot on a stack of a few books (last one narrower than robot wheel tread) to get the wheels off the floor while keeping it upright. Then you can play with objects in front of the laser and watch what the wheels are doing.
5. The code `figure(1); clf;` will keep a figure open and allow you to keep writing a new position for the “x” to it.
6. It’s sometimes useful to put a `clc` (clear command window) at the top but inside the main loop. Then, whatever you are printing will be in the same place and you get a rudimentary real time numeric display.

7. The MATLAB idiom `plot(-y,x);` will render data in robot coordinates such that robot x is up and robot y is to the left. That helps when you want your idea of "forward" when you walk while looking at your screen to be consistent with the robot convention. Flip both signs if you are walking backwards.
8. Test your motion algorithms in an uncluttered area, for your sanity, and to avoid wasting your time.
9. You should not need to use encoder data in this lab.