Introduction

As part of the Introduction to Robotics course, a hands-on project is important as a way to get a real-world feel of working with robots. In this project, a student can get some experience from the early stages of building the robot, through getting acquainted with sensors and motor control, and up to performing a complex task.

Platform

The robotic platform chosen is a ground based, 4 motorized wheels robot, with on-board sonar sensors and a marker based position service. The robot is controlled via an ESP8266 and Arduino development boards that use hardware pins to send commands to the motors and read the sensors; the controller also has WiFi connectivity and through the network it accepts commands to be sent to the motors, and in turn, forwards the values read from the sensors.
Project Task

The task to be done in this project is to develop code that implements a simple bug algorithm (full details in course lectures). The basic description is to have the robot drive in a straight line towards a goal position and reach that position. When the sensors detect an obstacle that does not allow continuing in a straight line, the robot should select a tangent direction and follow the contour of the obstacle until such time as the path towards the goal is clear.

Grading the performance of the implementation depends on the following factors:

1. Time to reach target. The shorter the better.
2. Obstacle avoidance. Hitting an obstacle induces a penalty.
3. Distance from obstacle when going around it. The closer the better (without hitting).

Since goal 1,3 seem to conflict, it is important to remember that sometimes the obstacle may require the robot to leave the camera area temporarily, so a strategy that tries to optimize speed without getting near obstacles may fail.

A test run consists of placing the robot in some location, and letting the algorithm run until the robot reaches the target, with a generous time limit of a few minutes (depending on the obstacle setup). Since the workspace is limited to around 3x2 meters, the distance to the target is limited. Obstacles are objects that are at least the size of the robot, so tiny objects do not count. Using boxes is a reasonable tool for training the algorithm.
Extra Credit

An extra task that can provide extra credit is the ability to let the robot traverse the environment, with a goal of creating a map of the obstacles. In this case, the output is a matrix of size 240x320 representing the workspace, where each cell contains 1 if an obstacle exists at that location, and 0 if not. Performance is measured using time to completion and a subjective evaluation of the resulting matrix.

Technical Details

Motor Control

Each side of the robot has two motors that always rotate with the same speed. The interface allows sending a speed command consisting of two values. One for the left side and one for the right. Each value is an integer in the range of $[-255, 255]$ where 0 is stop, 255 is forward full speed and -255 is backwards full speed. Since there is static friction, the motors will start to actually rotate at a value of around 60 (varies with each robot).

Sensors

Position

A ceiling mounted camera locates the green LED on the robot and sends a notification on the WiFi network containing a relative position. The values are approximately in centimeters, and relative to the center of the optical axis of the camera. This makes the X values range around $[-160, 160]$ and the Y values around $[-120, 120]$. The rate of position update is limited by the frame rate of the camera, and should be around 10-15 updates per second. Obviously the reading is noisy, but the variance is relatively small.

Obstacles

Obstacles are detected using the mounted sonar sensors. Each such sensor measures the distance to the object in front of it using transmission of an ultrasonic wave and measuring the returning wave. The sonar has a cone angle of about 15 degrees, and can measure objects at a theoretical range of 2cm - 200cm, but the effective range can vary, and should be tested on your robot. In addition, some cases may produce widely inaccurate results when waves are measured after being reflected on several surfaces.

Robot Interface

The serverless robot interface operates over UDP messages sent between the workstation, the robot and the camera.
Since the robot controllers cannot connect to an Enterprise wireless network (such as used in the Technion), a local wireless network is setup in the lab. The network SSID is the course number, and password will be given to students when they start working.

**Protocol**

After connecting to the wireless network, a UDP socket can be used to send drive commands to the robot. In order to send, the only information needed is the robot’s IP address and the port (which is 2777). The message is a simple string containing two integer values for the left and right motors, respectively. No end-of-line is needed.

Creating another socket that listens for UDP messages on the same port will produce incoming messages from the robot and camera. The robot reports 3 floating point values encoded as a string with space delimiter. The camera reports in the same format, but only 2 values for X,Y.

**Software API**

Protocol implementation is provided in two languages: C++11, Python.

The C++ implementation relies on the Boost\(^1\) library, and is tested on a Linux platform. The Python implementation uses version 2.7, but can easily be modified to 3.x

**C++ API**

The C++ interface consists of a single class shown in (1). The class will start a new thread that receives notifications and processes them. The only non-intuitive part is the format of the values in the `sense` method. These should be 5 values: `X Y D1 D2 D3` as described above.

**Algorithm 1 C++ interface class**

```cpp
#include <iostream>

class RClient
{
public:
    RClient();
    virtual bool drive(int left, int right) = 0;
    virtual bool sense(std::vector<double>& values) = 0;

    typedef std::unique_ptr<RClient> client_ptr;

    client_ptr connect_client(const char* robot_ip, const int port);

};
```

\(^1\)http://www.boost.org
Python API

The Python interface consists of a single module (udpclient) that implements functions very similar to those in the C++ interface. This module has a class called RClient. The following code snippet shows an example of usage:

```python
import time
from udpclient import RClient

r = RClient("192.168.1.151", 2777)
run.drive(80, 80)
print r.sense()
time.sleep(2)
run.terminate()
```

General Notes

Code

All supplied code is maintained on a git server, and copies of the repositories exist on the 3 workstations in the lab, under directory `~/is/rlgit/mars`

There are 3 main directories in the repository:

- python - containing the server code, and the python based client
- cpp - containing the C++ based client
- arduino - containing the firmware code for both the Arduino Nano v3, and the NodeMCU v1.0 (which contains the network information hard-coded)

LED

The LED on the robot is used for location and should be visible. It is also used for checking the network status. These are the possible behaviors of the LED:

- Constant on - Connected to server, good to go.
- Blinking twice each second - Connected to WiFi, but not to server. Probably server is down.
- Blinking once every 2 seconds - Failed to connect to WiFi

Batteries

- The batteries used are 2 cell Lithium Polymer, with 8.4V when full without load.
- It is generally a good rule of thumb to not let the voltage drop below 7V, as they can rapidly discharge and be destroyed.
• Only use a lithium polymer capable charger. During weekdays in the morning, we can use a fast charger to charge them for you. A regular charger will be available in the robots cabinet.

• Use the voltage meter once in a while to make sure the battery is not too empty.