Introduction to Robotics (236972)
Course Project

January 6, 2020

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 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 virtual sonar sensors and a marker based position service. The robot is controlled via an ESP8266 development board that uses hardware pins to send commands to the motors; the controller also has WiFi connectivity and through the network it accepts commands to be sent to the motors. In addition to the robot, there is a ceiling mounted camera which captures the scene, and provides information of position and orientation of the robot, along with information about obstacles.

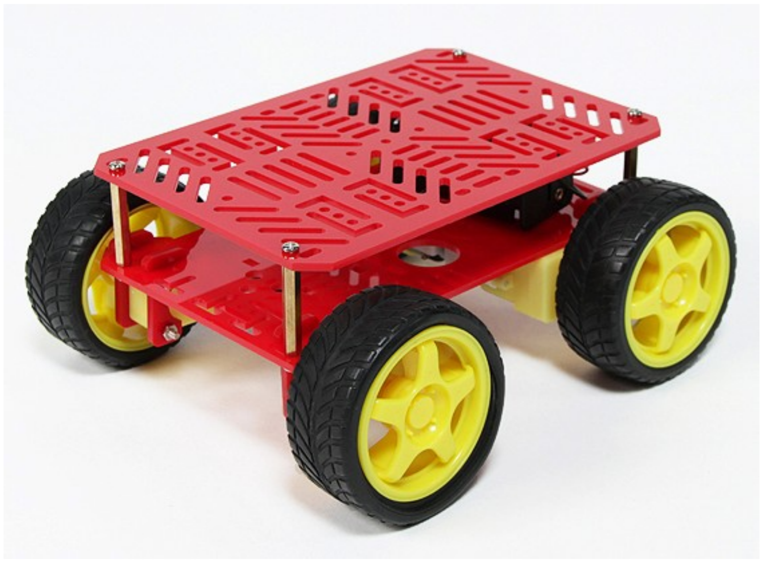

Figure 1: Base Robot Platform

Project Task

The task to be done in this project is to develop code that implements a simple trajectory planning algorithm (full details in course lectures). The basic description is to have the robot drive in the fastest way 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 employ some method to go around the obstacle until such time as the path towards the goal is clear.

The main grading factor is the time performance of the implementation. This means, lower time, higher grade. In addition to this, penalties will be awarded on each hit of an obstacle.

A test run consists of placing the robot in some location, and letting the algorithm run until the robot reaches the target and stops, with a generous time limit of a few minutes (depending on the obstacle setup). When the time limit expires, the run is considered failed. Since the workspace is limited to around 4x3 meters, the distance to the target is limited. Obstacles are segments of blue pool noodles, which will be available for training the algorithm. The scenario will consist of a known obstacle and additional unknown set of obstacles.

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 (image) of scale 1cm per pixel 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. Timing and evaluation of this task is separate from the main task.

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 $[-1000, 1000]$ where 0 is stop, 1000 is forward full speed and -1000 is backwards full speed. Since there is static friction, the motors will start to actually rotate at a value of around 250 (varies with each robot).

Sensors

Position and Orientation

A ceiling mounted camera locates the two LEDs on the robot and sends a notification on the WiFi network containing a position and orientation vectors. The position values are in centimeters (approximately), and relative to the center of the workspace (marked on the floor as 0, see Figure 2). The orientation is a direction vector indicating the foward direction of the robot. The rate of update is limited by the frame rate of the camera, and should be around 10 updates per second. Obviously the reading is noisy, but the variance is relatively small.

Obstacles

Obstacles are detected using the virtual sonar sensors. These are calculated using the ceiling camera and provide distance information for a set of angles around the robot. Each such sensor measures the distance to the object in front of it up to a certain range of about 1 meter. A typical setting would be 3 sonars, where one is facing forward, and two are left and right $45°$ or $90°$. The final sonar setup will be anounced later, and the algorithm should be somewhat robust to change of setup.

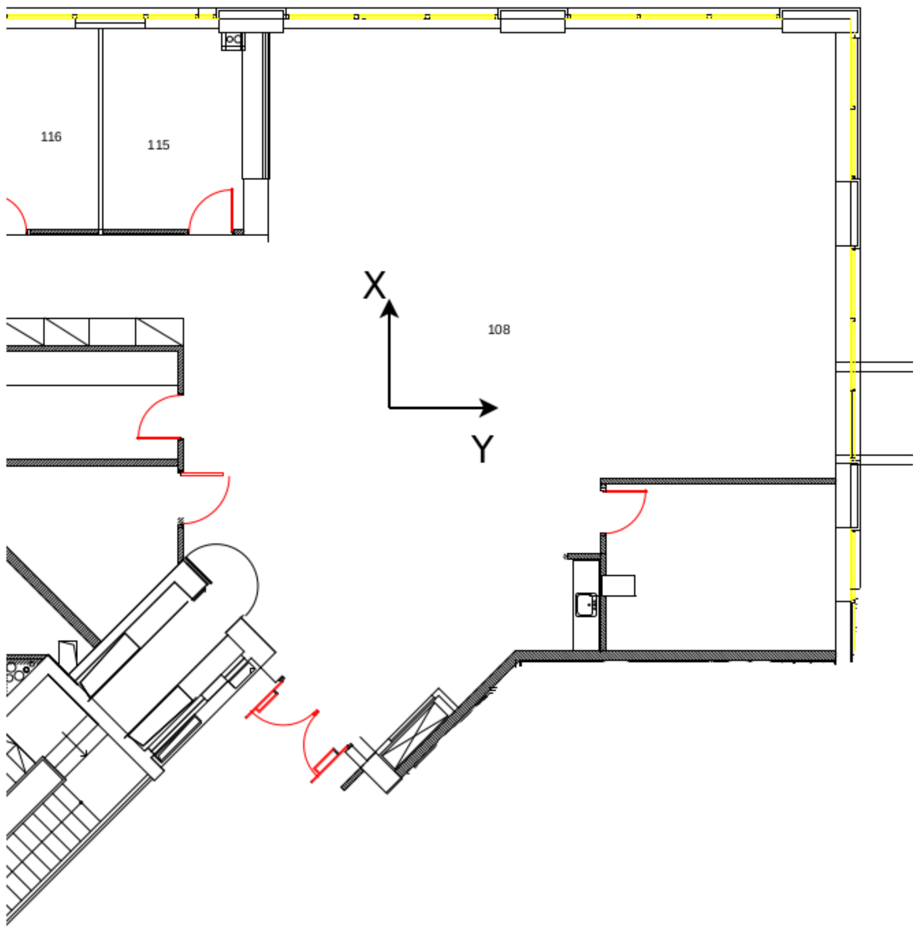

Figure 2: Lab coordinate system

Robot Interface

The 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 camera. The camera reports $4+n$ floating point values encoded as a string with space delimiter. The first 4 numbers are X, Y, D_x, D_y and the other n values are the sonar distance measurements (or a negative value if no obstacle is sensed).

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

```
class RClient
{
public:
  virtual ~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);
```

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:

[1]http://www.boost.org

```
import time
from udpclient import RClient

r=RClient("192.168.1.151",2777)
r.drive(480,480)
print r.sense()
time.sleep(2)
r.terminate()
```

General Notes

Code

All supplied code is provided in a zip file uploaded to the course site
There are 2 main directories in the repository:

- python - containing the server code, and the python based client
- cpp - containing the C++ based client

LEDs

The LEDs on the robot is used for location and should be visible. The green led is facing forward.

Batteries

- The batteries used are 2 cell Lithium Polymer, with 8.4V when full without load.
- Only use the supplied charger, which is available in the lab. A typical full charge should be enough for an entire day, if the battery is unplugged during development cycles.