Scribbler Bot Proposal

Mitchell Kember

SE 101

Dr. Morton

3 October 2014

Introduction

I propose to develop software that will enable the Scribbler Bot to map its surroundings. It will be called *Cartographer*, and it will include a web application (server and client) as well as a program to control the robot.

Description

Once the server is running, the user accesses the web application (remotely, if they wish) and starts the program by clicking the appropriate button. The robot begins to explore its environment and it detects obstacles using its sensors. It sends status messages to the web application console, and the map gets displayed as it is built.

The user can pause and resume the program, adjust the parameters of the program, and perform operations such as making the robot beep or take a photo. In addition, they can click on the map to instruct the robot to drive to an arbitrary position. (In order to do this, the robot must maintain an accurate estimate of its position within the map.) Finally, the user can save maps for later use, provided the robot is placed in the same initial position.

Use of robot features

The *obstacle sensors*, the *infrared sensors*, and the *stalling sensors* will all be useful for this project. The first two seem similar, but I suspect that one is more accurate than the other. The stalling sensors in the motors will be essential whenever the long-distance methods fail. If the robot is trying to drive in some direction but cannot move, there is almost certainly an obstacle in its way that should be added to the map.

Python modules

The part of the software controlling the robot will of course rely on `myro` [1]. The server will use a few of Python's built-in modules, namely `argparse`, `datetime`, `os`, `sys`, and `webbrowser`. We will also use a separately installed module, `gevent` [2]. This module provides lightweight execution units called Greenlets that greatly simplify the challenge of concurrency. Since the server must be able to respond to multiple requests concurrently while the main loop of the robot program is executing, the application clearly needs to be asynchronous. The `gevent` library is perfectly suited for this, and it is well-documented. I discovered it while searching for methods of using WSGI concurrently [3].

Software stack

Cartographer can be viewed as a stack of five levels: Client, Server, Controller, Program, and Robot. Their responsibilities and methods of communication are outlined in Figure 1. Figure 2 shows the web client as it appears in Safari in its current design.

Design challenges and risks

There are two main challenges in this project. First, we must devise an exploration algorithm for the robot. The naive approach would be to simply scan up and down in one-dimensional strips along the room, but this would be slow and not at all reminiscent of intelligence. We can minimize the risk here by placing the robot in an artificial environment, such as a maze, rather than a room filled with strangely shaped objects (which would in any case have to be small enough to accommodate the Bluetooth range). Second, the robot must know its position within the map at all times. This calculation will depend critically on the consistency of the motors: we must know that driving for x seconds will result in a displacement of y metres. In any case, the position error will grow over time—unless we include error correction mechanisms, such as periodically returning to an identifiable "home position."

Time estimate

I have mostly completed the Client, Server, Controller, and the base Program. These are in no way specific to *Cartographer*, so they can still be used if this proposal is not selected. If it is, then most of our time will be spent developing the mapping Program. I predict that the exploration algorithm, the refinement of the position calculations, and the further development of the web application will take 40 hours, 60 hours, and 15 hours respectively. I believe our use of GitHub (http://github.com/mk12/scribbler-bot) and Trello will help us to stay organized so that we can use our time to its greatest potential.

References

[1] "Myro reference manual," http://wiki.roboteducation.org/Myro_Reference_Manual. Accessed October 1, 2014.

[2] Bilenko, D. "Gevent: A coroutine-based network library for Python," http://gevent.org. Accessed October 1, 2014.

[3] Hellkamp, M. "Primer to asynchronous applications," http://bottlepy.org/docs/dev/async.html. Accessed October 1, 2014.

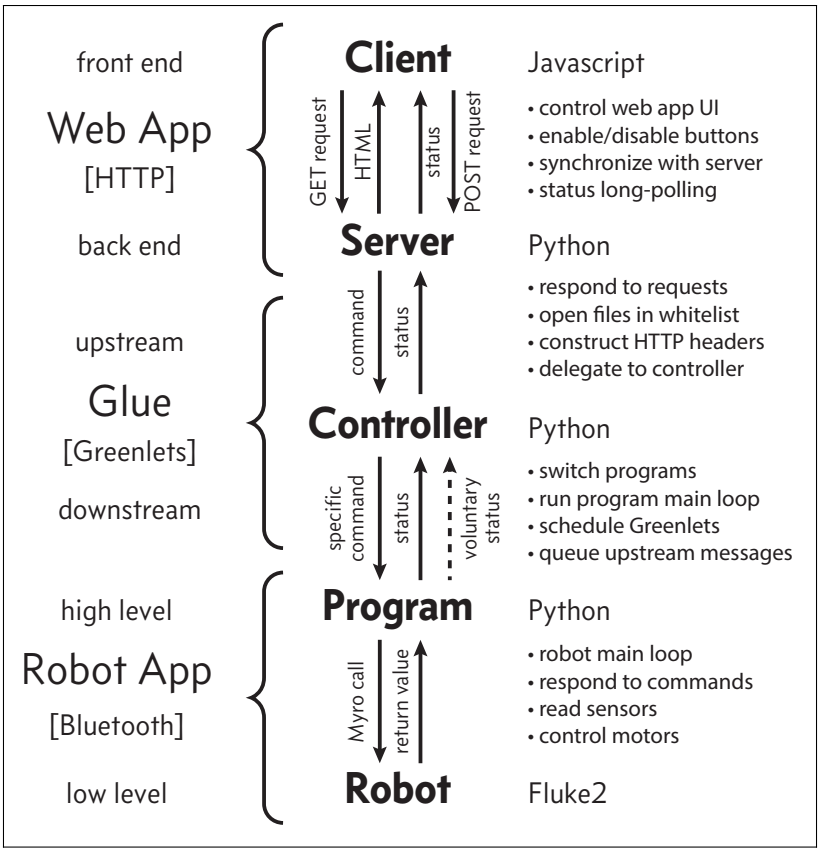

Figure 1 The software stack of *Cartographer*

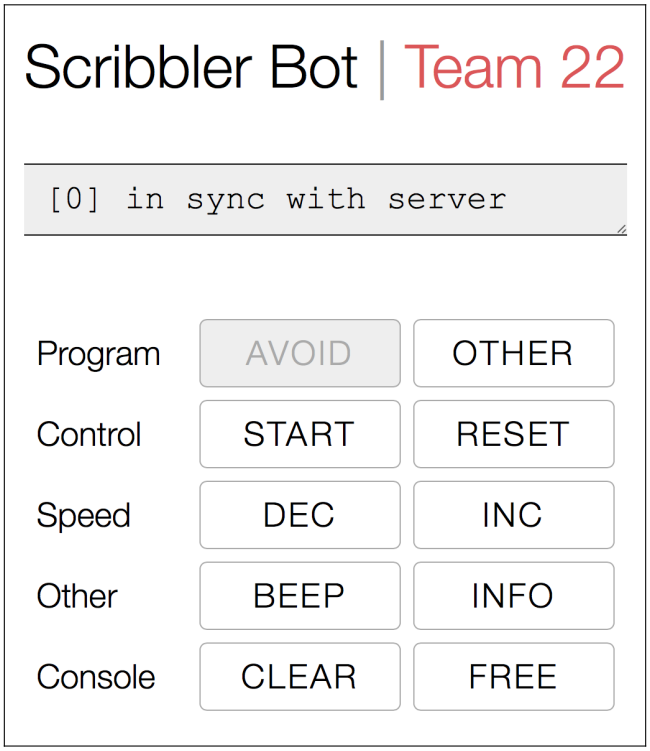

Figure 2 A screenshot of the web application