Massachusetts Institute of Technology

Robotics: Science and Systems – Spring 2005

Lab 2: The Carmen Robot Control Package, and an Example Application: Visual Servoing

Tuesday, September 13^{th}

Objectives

Last Thursday, your team recovered your hardware from last semester, and ensured that all parts were working. Over the next two weeks, you will recover your software, and practice individual and team software development skills before tackling the grand challenge. Once again we'll be using the robot control package Carmen. You should think about how to abstract implementation-dependent details. For example, how would you generalize your implementation to another robot, in another environment?

This is the last formal lab you will have for R:SS II. Over the next two weeks, we will present you with the structure to get you re-acquainted with the basic problems of sensing and acting in the real world. After this lab, you will be setting your own milestones.

Your objectives in this lab are to:

- Re-familiarize yourself with the structure and use of the Carmen robot control package;
- Re-familiarize yourself with on-line digital image acquisition and image processing operators including blob detection, blob size and centroid estimation and illumination calibration;
- Re-familiarize yourself with high-level decision making and motion planning;
- Implement an auto-calibration system, that will learn new calibration parameters for different coloured balls;
- Implement a simple motion planner, that will perform a tour of a pre-specified arrangement of balls;
- Document your implementation;
- Provide your implementation and documentation for another team to use;
- Test another team's implementation and documentation.

1 Re-familiarize yourself with your Robot

For the next two weeks, to minimize network problems, you will be running your robots tethered to the network. Your robot is configured to have two network names:

- Tethered ethernet: **rss-x-wired.mit.edu**. If you are team 2, your tethered robot will have hostname `rss-2-wired.mit.edu`. The IP address for robot X is $192.168.1.200 + X$, so `rss-2-wired.mit.edu` would have IP address $192.168.1.202$. When your robot is tethered, it is not directly accessible from the outside world.

- Wireless ethernet: **rss-x.mit.edu**. If you are team 2, your wireless robot will have hostname `rss-2.mit.edu`. The IP address for robot X is $18.34.0.XXX$, but there is no pattern matching group number to IP number. When your robot is wireless, you can connect to it directly from the outside world.

We have not created individual user accounts on the robots; like last year, each robot has a generic `rss-student` account with the same password.

2 Re-familiarize yourself with Carmen

Before beginning to control your robot with Carmen, make sure that your OrcBoard is connected to your SBC with a serial cable, and the connection is secure at both ends. Make sure your camera is plugged into your SBC. Make sure your that your robot is up on blocks (i.e., that its wheels are clear of the floor or table).

If you are unfamiliar with Carmen, a brief tutorial in how Carmen was used in R:SS I is provided at the end of this handout. Additionally, the *Guide to the Use of the Carmen Mobile Robot Control Package Within RSS* is available on the RSS course page, and more Carmen documentation is at `http://www.cs.cmu.edu/~carmen`.

Carmen has had some revisions since last year; you can get the latest version of the software using subversion:

```
% svn checkout svn+ssh://rss-student@rss-sun-la:/srv/svn/repository/rssII
```

Note that if you wish to access the repository from outside the lab (i.e., from a non-RSS machine), the hostname changes from `rss-sun-la` to `rss-hangar@mit.edu`, as in

```
% svn checkout svn+ssh://rss-student@rss-hangar.mit.edu:/srv/svn/repository/rssII
```

from your laptop on the MIT network.

2.1 On the robot

1. Check out a copy of the R:SS II repository on your robot (`rss-x-wired`).

   ```
   % svn checkout svn+ssh://rss-student@rss-sun-la:/srv/svn/repository/rssII
   ```

2. Inside the `rssII/trunk` directory, you should see 5 directories:

   ```
   carmen bin lib include Vision
   ```

   Of course, all of the files will show up in both places (the robot and your workstation), but some files will only be useful in some places. You should run the robot control programs on the robot itself. Typically you will run the GUIs only on your workstation or laptop.

   Inside `bin/` reside the four main Carmen binaries for robot control, namely,

   ```
   message_daemon param_daemon orc_daemon robot_central.
   ```

3. You should be able to start these programs **on the robot** as you did last semester by running each one in a separate xterm that is connected to your robot using SSH.

   (More instructions exist at the end of this document, but if you run the commands in the order given above, with no arguments, you should be on your way.)

2.2 On your workstation

1. Check out a copy of the R:SS II repository to your local workstation (`rss-sun-x`).

   ```
   % svn checkout svn+ssh://rss-student@rss-sun-la:/srv/svn/repository/rssII
   ```

2. Inside the `Vision` directory **on your workstation**, you will discover java class files, including the `RobotGUI` class from last year. Compile the class files using `make`, and run the `RobotGUI` using

   ```
   % java -cp ../carmen/Carmen.jar:. RobotGUI
   ```

3. **You must tell the `RobotGUI` (running on your local workstation) where to find your robot on the network. You should either edit `RobotGUI.java` to give your robot a default IP address, or provide your robot's IP address on the command line, as in:**

   ```
   % java -cp ../carmen/Carmen.jar:. RobotGUI 192.168.1.202
   ```

4. Once `RobotGUI` has connected to the robot, you should see the robotGUI display, and a blue window where the camera display would be (since you are not yet running the `camera_daemon`).

5. Control your robot's wheel motions using the keyboard ('I' to move forward, 'J'/'L' to turn left, right, etc.). Do the wheels turn the right way? When in contact with the floor, will the wheels turn the robot the same way as it rotates on the screen? If not, are your motors connected to the correct respective ports? Are the motor encoders connected to the correct respective ports?

3 Re-familiarize yourself with image acquisition and processing

Inside `bin`, you will discover the main Carmen binary for image acquisition, `camera_daemon`. Start the `camera_daemon` on the robot. Did `camera_daemon` start? Is your camera plugged in? Is it complaining about permissions on `/dev/video0`? If so, consult a TA. Are the `pwc` and `pwcx` modules installed? (See the appendix.) If not, consult a TA for help running the `install_quickcam_drivers.sh` script.

At this point, you have a choice: you can either recover your code from last semester, and run your own `VisualServo` module, or you can run the "solution" `VisualServo` module we have provided for you in the `Vision` directory. **Remember to specify your robot's IP address to `VisualServo` as well.**

You can invoke `VisualServo` (either the provided solution or your own) as:

```
% java -cp ../carmen/Carmen.jar:. VisualServo
```

you should see the camera image in the right pane of `RobotGUI`. If you hold up a ball of the right colour, the displayed image should contain labelled pixels.

At this point, you need to start modifying code or writing your own. You should **not** check code into the `rssII` repository, as this will corrupt everyone else's code. We have provided you with your own repositories, that you can check your modified code into.

Check out your own group repository using:

```
% svn checkout svn+ssh://rss-student@rss-sun-la:/srv/svn/repository/group-x
```

Now, copy the files you will be changing (e.g., `VisualServo.java`, `RobotGUI.java`) into your local repository.

4 Implement an auto-calibration system

Consider what must happen if the colour of the ball changes, Either because the right colour ball is missing today, or because the lighting has changed and now the same colour ball *looks* different to the robot's camera system. We need to be able to re-calibrate our image processing module quickly.

Write a java program that records a file of colour names mapped to HSV triplets. The program should do the following:

1. Wait for keyboard input
2. Take either a colour name or the word "`end`" from the keyboard
3. If a colour name is received, assumes that a ball is being held in front of the camera. The current image is processed, the dominant HSV colour (mode of the histogram) is recorded, and labelled with the colour name. The program then waits for more input (step 1).
4. If the word "`end`" is received, the program should prompt for a filename, write the list of entered colours and associated HSV triplets to the specified file, then exit.

Note that we have not specified the file format. That's up to you.

Modify the `VisualServo` program (either your own, or the one provided), to load your file of colour names and HSV values, and then visual servo towards any ball of a learned colour.

5 Implement a simple motion planner

Now, imagine wanting to take a tour of multiple coloured balls laid out on the hangar floor. Simple visual servoing is unlikely to solve this problem, unless the robot can see most of the balls from most of the floor. If, however, we move the balls out of visual range of one another, then we have to actually perform motion planning. In this lab, we won't ask you to perform sophisticated obstacle avoidance, although you should be able to do so.

Write a java program that accepts a list of coloured balls and their locations relative to the start location of the robot:

1. Set up your robot at some location on the hangar floor
2. Set up 4 coloured balls around the robot, separated from each other and from the robot by a metre or two
3. Assuming the robot is at $(0, 0)$, and facing along the positive x axis, measure the (x, y) positions of each ball relative to the robot
4. Record the colour and position of each ball in a text file
5. Write a java program that

(a) Reads the text file **and** colour name/HSV calibration file

(b) Turns the robot to face the first ball

(c) Drives the robot until a ball of the appropriate colour is in visual range

(d) Begins visually servoing to the ball, until the ball is centred in the visual field

(e) Repeats the tour to the second ball, until all balls have been "surveyed". If the current ball obstructs the next ball, do *not* drive through the ball but drive *around*.

6 Document your implementation

A major deliverable of this lab is to be able to document your code sufficiently well that someone else can use it. For consistency, we ask you to implement the following documentation procedure:

- You should develop a complete, itemized set of instructions on how to operate your implementation.
- You should have a complete set of documentation on all classes. Document your source code **inside** the code, then use `javadoc` to auto-generate HTML documentation. Then incorporate this HTML in your wiki.
- You should have complete documentation of all your file formats (e.g., calibration files, motion plan files, etc.) in your wiki as well.
- You should have complete documentation on how to retrieve your code from the repository.

A word on source control: We are supplying you with code via the `rssII` repository – this repository is *not* writable by you. Each group has their own repository, which is currently empty. You can check out your repository using

```
% svn checkout svn+ssh://rss-student@rss-hangar.mit.edu:/srv/svn/repository/group-x
```

(where `x` is your own group). Of course, you can add code to your repository with `svn add` and `svn commit`.

7 Testing another's implementation

On Thursday, September 22nd, you will stop development of your own implementation, and spend the lab time testing another team's implementation. Each team n will be tested by team $((n+1) \ \% \ 5) + 1$ — that is, team 5 will test team 4, team 1 will test team 5, team 2 will test team 1, and so on. You should begin by reading the wiki of the team you are testing, to learn how to check out and run their code. In your own wiki, you will document how well you were able to run the code. You should make concrete, **constructive** suggestions for improvements to both the team's implementation and their documentation. (Make sure to be constructive; gratuitously negative comments will damage your own team's grade, not the other team's.)

Time Accounting and Self-Assessment.

This lab can be a demanding one in terms of time. You must therefore learn to parallelize development. For example, you may wish to generate all necessary class files as empty, with stubbed out methods, etc. *first*. You may then wish to split into two groups, one group of two team members developing the auto-calibration code and two team members developing the motion planner simultaneously. If you choose to parallelize like this, you should consider generating the documentation on class files and file formats *first* before beginning implementation. That is, we suggest you agree on, and document, your code specification before implementing it. You may then wish to assign team members different tasks in terms of documentation, testing, etc..

After preparing your report on the success of running your partnered team's code, return to the Time and Assessment pages of your individual logbooks. Tally your total individual effort there, including time spent writing up your report. Answer the "Self Assessment" questions again, post-Lab. Add the date and time of your post-lab responses. *After you have done this individually, please also add a sentence to your team's Lab report stating the individual number of hours each member devoted, and the total number of person-hours your team devoted, to the lab, including T/R lab hours.*

8 Appendix

8.1 The Carmen Robot Control Package

Carmen is a distributed collection of modules (technically, processes, each running in a separate address space) that together implement autonomous robot control, and support user operations such as remote modification and monitoring of robot internal state. Carmen modules communicate through a networked, anonymous "publish/subscribe" mechanism, in which the only supported communication mechanisms are to publish data of interest to other subscriber modules, and/or subscribe to receive notification of data published by other modules.

A robot control system built on top of the Carmen toolkit will ordinarily invoke at least three processes to support autonomous operation:

- The `message_daemon` process provides inter-process connectivity to all Carmen modules, ensuring that modules may publish any data, and that modules may subscribe to any data type (and be notified whenever new values of that data type become available).
- The `param_daemon` process provides a central repository for robot configuration parameters (e.g., odometry scale factors and biases, maximum velocity constraints, controller gains, etc.). `param_daemon` reads the `carmen.ini` file at startup and publishes its contents to any interested subscribers via `message_daemon`. `param_daemon` also supports changing (and re-publishing) some parameters during robot operation.
- The `robot_central` process (`robot_central`) integrates the published outputs of various sensor modules (including, but not limited to, the `orc_daemon` and `camera_daemon`), performing time-stamping and odometry-stamping of each captured sensor measurement and publishing the stamped values.

  `robot_central` also provides local response (or "reflex") behaviors for current or imminent collisions, e.g., bump sensor activation or fast-closing sonar returns.

Because we use the OrcBoard for motor and sensor management, and because in this lab we are developing a semi-autonomous (i.e., human-monitored) visual servoing capability, our robot system will invoke four additional Carmen modules:

- The `orc_daemon` process abstracts away the details of controlling the robot into high-level, body-relative velocity commands, and makes current odometry available to any subscribers. `orc_daemon` will also pass bump-sensor and sonar state updates to `robot_central` in future labs. `orc_daemon` is a Carmen replacement for `orcd`.
- The `camera_daemon` process abstracts away the details of managing the camera state, including any captured images. `camera_daemon` publishes the most recently acquired camera image as a 2D array of RGB pixels.
- The `VisualServo` process analyzes the most recent captured image, and publishes robot velocity commands intended to size and position the detected visual target (if any) in the robot camera's field of view.
- The `RobotGUI` process provides a graphical user interface (GUI) with which one or more users can remotely command the robot, and observe and modify its internal state.

9 Download and Incorporate the New Source

We have placed the Carmen binaries in the `bin` directory under `rssII`.We have placed the Carmen class library (Carmen.jar) in the `/rssII/trunk/carmen` directory. In this directory we have also provided the Carmen java source. You should not need to compile this source, but will want to familiarize yourself with the classes and interfaces.

We have placed java files (`RobotGui.java`, `VisualServo.java`, etc) in the `trunk/Vision` directory. We have also provided a makefile for command line compilation.

There are a few critical steps in installing Carmen. It is important that you follow our recommended source directory structure and build environment. If you are unsure about a step, consult a TA.

First you will configure your Sun Workstation environment:

1. Check out the `rssII` repository using

   ```
   % svn checkout svn+ssh://rss-student@rss-sun-la:/srv/svn/repository/rssII
   ```

2. Check out your own code repository using

   ```
   % svn checkout svn+ssh://rss-student@rss-sun-la:/srv/svn/repository/group-x
   ```

   Copy all the files from `rssII/trunk` to `group-x`.

3. The next step is to set your bash environment up correctly. In the directory `~/rssII/trunk/conf/` is the file `dot_bash_profile`. Copy this to `~/.bash_profile` and edit the `$RSS_GROUP_HOME` variable. (Or merge it with your existing `~/.bash_profile` if you want to keep your existing settings.) Open a new shell to use the updated `.bash_profile`.

4. Verify that you can compile the Vision source by: `cd ~/group-x/Vision; make`.

Now proceed to configure your SBC environment.

1. Start up the SBC.

2. SSH into the SBC with `ssh rss-student@rss-X`.

3. Check out the `rssII` repository using

   ```
   % svn checkout svn+ssh://rss-student@rss-sun-la:/srv/svn/repository/rssII
   ```

   as before.

4. Check out your own code repository using

   ```
   % svn checkout svn+ssh://rss-student@rss-sun-la:/srv/svn/repository/group-x
   ```

   Copy all the files from `rssII/trunk` to `group-x`.

5. Incorporate the default profile `~/rssII/trunk/conf/dot_bash_profile` into your `~/.bash_profile` as before. Close and reopen the SSH session to use your updated `~/.bash_profile`.

6. Install the drivers for the QuickCam with the script: `~/rssII/trunk/conf/install_quickcam_drivers.sh`.

7. Plug in the camera to the SBC's USB port. The QuickCam should be auto-detected, and the `pwc` module should load automatically (courtesy of `\etc\init.d\hotplug`).

8. Manually load the `pwcx` module with `~/rss/rss_groupN/conf/load_quickcam_modules.sh`.

9. Verify that the required modules are loaded by doing `sudo lsmod | grep pwc*`. (Ask a TA for the required password.) After this, you should see:

```
pwcx                   89056   0  (unused)
pwc                    47536   0  [pwcx]
videodev                6464   1  [pwc]
usbcore                62924   1  [audio pwc usb-uhci]
```

   If you do not see the above module list, run `~/rssII/trunk/conf/reset_hotplug.sh`, and try again.

10. Open five shell windows and in each, SSH onto your SBC. Here you will run five Carmen binaries, each compiled from C source. You will not need to modify or rebuild any of these binaries for this lab, although this is possible if you really want to (ask the staff). Within each shell, start one foreground Carmen process, respectively:

    `message_daemon`

    `param_daemon`

    `orc_daemon`

    `camera_daemon`

    `robot_central`

    You must start `message_daemon` first, then `param_daemon`, but the rest can be started in any order. Also, `param_daemon` must be started from within `bin`.

11. Now verify that the QuickCam is working. SSH with `ssh -X rss-student@rss-X`, and run `camera_view`. You should now see the camera video. If the QuickCam is not working, ask a TA for help.

12. On the Sun, in RobotGUI.java and VisualServo.java, modify the line

```
Robot.initialize("VisualServo", "rss-X");
```

    to use the hostname of your SBC. Recompile the source.

13. Now on the Sun, start the remaining Carmen processes from your `group-n/Vision` directory:

```
java -cp ../carmen/Carmen.jar:. VisualServo
java -cp ../carmen/Carmen.jar:. RobotGUI
```

    The RobotGui should show you an iconic overhead view of the robot, and a reduced version of the camera image next to it. Congratulations! You are ready to start implementing your solution.

10 Use Carmen to control the robot

RobotGUI subscribes to odometry-stamped camera data (i.e., images) of class VisionImageMessage published by VisualServo. In one pane, RobotGUI draws an icon that represents the current robot state.

You can drive the robot around using RobotGUI. See RobotGUI.java for the keybindings that drive the robot. Take the robot for a spin. Note: the robot origin is pinned to the center of the robot pane. The robot is drawn with an orientation or "attitude" equal to its signed orientation with respect to its orientation at time 0 of the current robot session.

The other RobotGUI pane shows the most recent camera image. Wave an object in front of the camera and see the most recent image change.