

# Robotics: Science and Systems II – Fall 2006

## Lane Following and Obstacle Avoidance

**Begins: Thurs 9/28/06, 2pm**

**Due: Thurs 10/12/06, 12pm**

## 1 Objectives and Lab Overview

In this lab, you will choose to focus on either the cameras, or the laser scanner. We highly recommend working in groups.

### Week 1

Students working with the cameras are responsible for writing modules that are able to detect lanes of a road, and that generate waypoints that cause the vehicle to follow the road.

Students working with the laser scanner are responsible for writing modules to detect obstacles in the way of the robot, and a simple path planner that, given a list of obstacles and a waypoint, steers the robot around the obstacles to reach the waypoint.

### Week 2

In the second week, camera and laser teams must integrate their code so that the robot is able to successfully follow a road while avoiding obstacles. **Do not underestimate the difficulty of this!** The two teams must agree on a messaging protocol, and modify their modules to correctly implement this protocol. It's not a bad idea to choose a team with which to integrate during the first week of this lab, and to design an interface early on.

## 2 Programming Environment

The software for this lab builds upon the software used in the previous lab. We recommend that you do a fresh subversion checkout in a new directory, as many files have changed significantly.

```
svn co --username rss-student https://svn.csail.mit.edu/rss-dgc lab2
```

We will not have the resources to help you configure a personal workstation. This lab introduces a few new dependencies. Specifically, it requires the Intel Integrated Performance Primitives version 5.1 (5.0 is too old), and recent versions of the `libdc1394` and `libraw1394` development libraries.

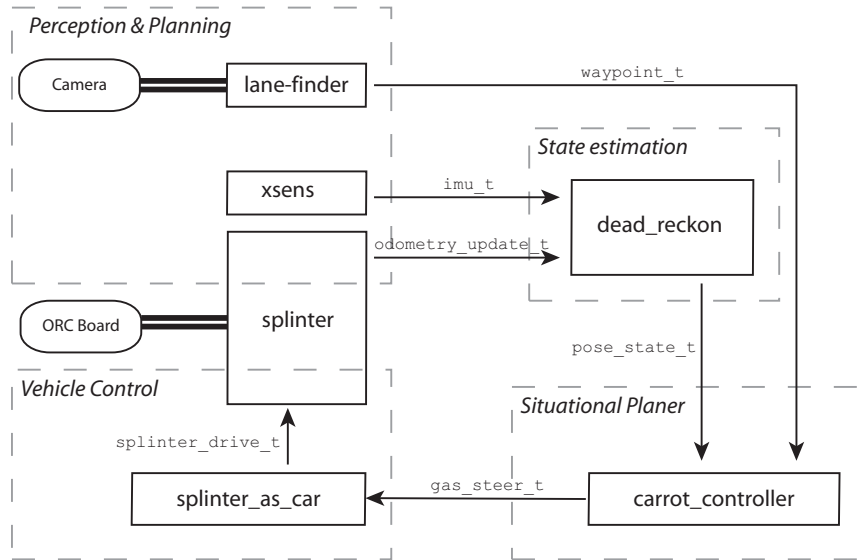
## 3 System Diagram

You are free to implement a solution in any way you see fit, but here are some suggested diagrams that you're welcome to follow.

# 6.142 / 6.897 / 16.401

## Lab 2 Suggested System Diagram - Lane Follower

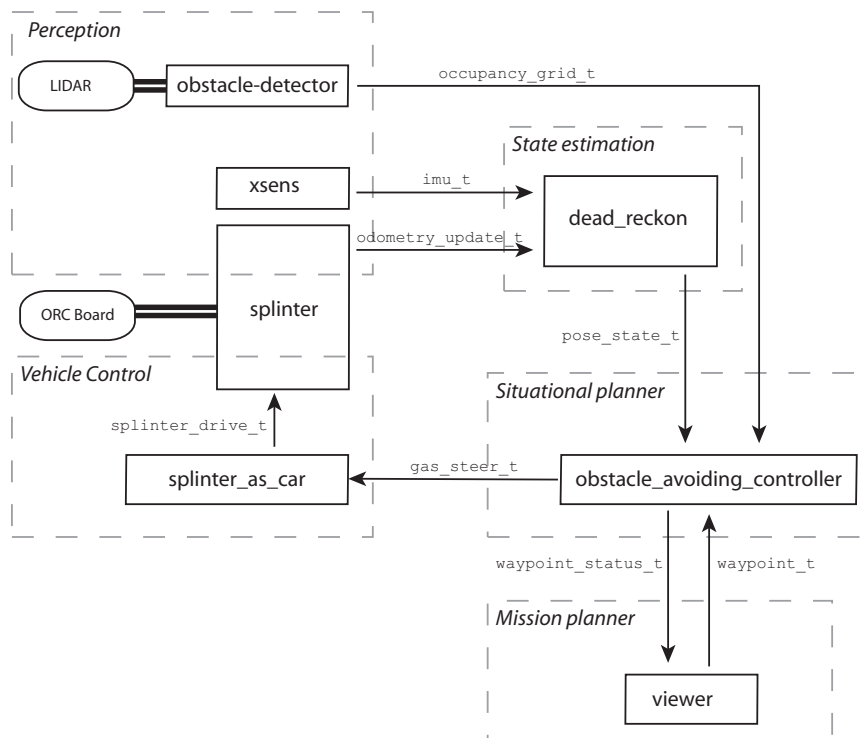
Sep 27, 2006

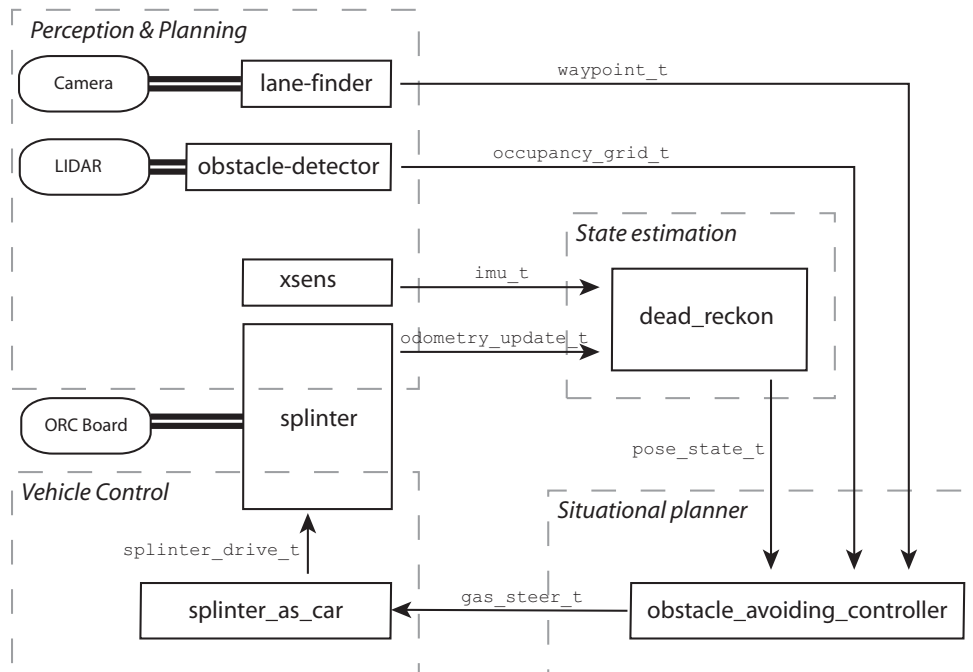


# 6.142 / 6.897 / 16.401

## Lab 2 Suggested System Diagram - Obstacle Avider

Sep 27, 2006





## 4 Build directory changes

In the previous lab, each module would build its executable file to a directory like `software/src/viewer/viewer`. We have changed the Makefiles so that all executables should now build to the `software/bin/` directory.

## 5 procman

In the previous lab, each individual process had to be started manually, by opening a new terminal and typing in each command separately. By now, you're probably sick of having to do this, as it's slow and easy to forget which processes need to be started. We've written a process manager that reads in a configuration file, executes each command listed in the file, and redirects all of their outputs to a single terminal. For example, to run all the modules needed for the simulated carrot controller in Lab 1:

```
# cd software/bin
# ./procman ../config/pm-carrotsim.txt
```

You may create your own configuration files and use them with procman. Hopefully, this will ease some of the pain of testing and debugging.

## 6 Demo

To run the lane-following demo, use the configuration file `pm-lanedemo.txt`

```
# cd software/bin
# ./procman ../config/pm-lanedemo.txt
```

To run the obstacle-avoiding demo, use the configuration file `pm-obstaclesdemo.txt`

```
# cd software/bin
# ./procman ../config/pm-obstaclesdemo.txt
```

At the present, we do not yet have a demo for the integrated obstacle-avoiding lane-follower, but this should be added shortly. When this happens, we will provide an update via subversion, and a configuration file for that demo.

## 7 Lane Following

At a bare minimum, your vehicle should be able to follow a lane marked by two bright yellow solid strips of tape laid down on the ground. Handling intersections is extra credit.

You are free to implement any lane following technique you want. A simple one that will work under tightly controlled lighting and environmental conditions is now described.

Calibrate the camera to the road surface by picking Hue, Saturation, and Value values and thresholds. Assign these to `target_h`, `target_s`, `target_v`, and `thresh_h`, `thresh_s`, `thresh_v`

```
while True:
    Capture an image frame
    Convert the image from RGB colorspace to HSV colorspace

    Create a blank binary img road_img

    For each pixel x, y with color h, s, v in the image:
        if h, s, and v are within the range defined by the calibration:
            road_img( x, y ) = 1
        else:
            road_img( x, y ) = 0

    Run a connected components clustering algorithm to cluster the marked
    pixels in road_img into discrete clusters.

    Pick the two largest clusters, and assume that they correspond to the
    lane markers.

    Find the geometric center cx, cy of the two lanes.

    If cx is on the left hand side of the image, then steer left. If cx is on
    the right hand side of the image, steer right. Otherwise, go straight.
```

There are many problems with this technique that will prevent it from succeeding in real world conditions. Dramatic lighting changes, poor calibration, other objects in the scene with the same color as the road, road intersections, damaged roads, sharp curves, are all factors that may contribute to the failure of this algorithm.

For extra credit, implement a technique that provides robustness against one or more of these situations, and describe your technique. For example, to avoid misclassifying objects with a similar color as the road, you can fit a 2nd or 3rd order polynomial to each cluster using a robust fitting method such as RANSAC. If there is no good fit, then the cluster may be rejected and the next cluster considered. Another extension might be to compute the extrinsic calibration between the camera and the robot, use this calibration to infer the 3D layout of the road, and follow the center of the road with control technique better than bang-bang control. If you are interested in any of these extensions, talk to the TAs first.

To help you in this task, we are providing you with source code for computing a connected-components clustering of pixels. Please see `software/src/rss-lane-follower/connected_components.h`

For an example on how to use it, take a look at `conn_comp_example.c`

## 8 Obstacle Avoidance

At a bare minimum, your vehicle should be able to navigate to a waypoint while avoiding obstacles, or send a waypoint status message indicating that the waypoint is not reachable.

You are free to implement any obstacle avoidance technique you want. A simple one that will work under reasonable circumstances is now described.

given a waypoint  $W_x, W_y$

create an occupancy grid, where  $occupied(x,y) = 1$  means that  $x, y$  is not traversable, and  $occupied(x,y) = 0$  means that the robot may successfully traverse that point.

while True:

    obtain a laser scan from the LIDAR. Use this scan to fill in the occupancy grid. For every occupied grid point  $(O_x, O_y)$  dilate the occupancy grid at that point so that  $occupied(O_x + dx, O_y + dy) = 1$ , where  $(dx^2 + dy^2) \leq robot\_radius^2$ .

    If  $occupied(W_x, W_y)$ :

        transmit an error: "waypoint is not reachable"

    Use A\* or Dijkstra's shortest-path algorithms to compute the shortest path from the current position to  $W_x, W_y$ .

    Steer the robot along this path.

This technique should perform reasonably well in real-world conditions, but much of its performance depends on the implementation. For example, computing an actual smooth trajectory may take some effort. Additionally, it is important to consider the vehicle dynamics when computing a motion path. The vehicles we use are not capable of sidestepping/strafing, and thus must either back up, make a large loop, or incorporate lookahead-planning in order to travel sideways.

## 9 To Hand In

Your report should include a detailed description of your algorithms (with pointers to the corresponding code), an explanation of how you implemented them, and the results of running them.

To handin your report and code, create a tarball named `6142-lab1-lname1-lname2.tar.gz`, where `lnameN` is the last name of student N in your group. Be sure to do a `make clean` before you handin (try not to email us with 10 MB tarballs).

Email your tarball to: `rss-tas@csail.mit.edu`