CMPEN 497 – Humanoid Robotics: P3 Instructions

Introduction

The purpose of this project is to implement visual servoing on the TonyPi robots. We'll be using Haar cascade classifiers to do face detection, and the robot will be constantly updating its head servos to point the camera at a detected face. To handle the movement, we'll be using PID controllers. There's also some multithreading going on, but all of that is already handled for you.

Step 1: Getting the Files

The first step to this project is to download the relevant files from Canvas. If you want to get the face detection worked out on a laptop or desktop before attempting to do it on the robot, you can use the computer version of the starter code which doesn't require any of the robot libraries. However, you will at a minimum need the Haar classifier XML file regardless of which version you're working with. Once you begin working on the robot, you will also need to have the PID controller file. Download the files and put them all in the same working directory.

Step 2: Face Detection

My personal recommendation for a starting point is to get face detection working, that way you can have something to track once you start working on the PID controllers. To do face detection, you'll first need to load the Haar cascade classifier into the variable named "classifier" in the starter code. You should look at the OpenCV documentation on loading cascading classifiers from a file.

Once you have the classifier loaded, you'll need to do the detection within the face_detection() function. You should only have to change one line in this function as everything else is provided. Refer to the OpenCV documentation for performing object detection using cascading classifiers. You will need to select appropriate values for the scale factor and the minimum number of neighbors parameters in the function you will be calling. You can experiment with other parameters, but these two are the primary parameters to consider. You'll see that there are tradeoffs between speed, accuracy, and strictness. Experiment until you reliably have face detection working.

You may also notice that there is some extra logic in the robot file. This logic scales down the input image by a factor of ½ on each axis to make classification faster and then scales the output results back up to the size of the original input. With this logic, in my working copy I was able to go from 2-3 detections per second to 5-10 detections per second which helped with the visual servoing. You'll likely need to change your parameters a bit.

Step 3: Utility Functions / Choosing a Face to Track

The next step is to choose which of multiple detected faces to track. These can be found at the bottom of the helper function section. Two possible options already have function stubs for you to fill out. These options are the largest detected face and the detected face whose center is closest to the center of the image. You'll need to fill out these function stubs, and you may consider other methods. Here's a list of possible methods:

1. Largest detected face
2. Face closest to center of the image
3. Face closest to last previously tracked face
4. Some other criteria you come up with

Additionally, you'll need to figure out what to do when a face isn't detected. The default option is to simply not update anything. Another possibility is to create a fake detection at the middle of the image so the PID controllers can update using that data.

Step 4: PID Controller Setup

Once you're able to detect and select faces, you'll need to start worrying about the motion. The first part here will be to create two PID controller objects, one for the pan servo and one for the tilt servo. A PID controller can be created with "PID.PID(P=value, I=value, D=value)" where each "value" is a placeholder for the corresponding PID controller parameters. You may want slightly different parameters for each controller, but I personally find that it's fine to use the same values for both controllers.

A good starting set of parameters for the PID controllers would P=0.5, I=0.0, and D=0.0

With the PID controllers set up, the next thing to consider is resetting them in the reset() function. A PID controller named "ctrl" can be reset with "ctrl.clear()" but be aware that this also resets the PID controller's integral threshold value. So if you use the thresholds (also called "windup guards") you'll need to reset those after calling the clear() function.

Step 5: Doing Movement

With the PID controllers set up, you can begin to do the motion. For a PID controller called "ctrl" and using some placeholder values, the block of code to update it would be:

```
ctrl.SetPoint = target_value
ctrl.update(detected_value)
delta = int(ctrl.output)
motor = clamp(motor + delta, motor_bounds)
```

Once you've updated the values, you'll need to call the set_pan() and set_tilt() functions using your current servo values. You may find that you get better results by slowing down the motor timing from 100 to 150, 200, or 250. You can update the move_time variable as you see fit, but it should never be set below 75.

You may also wish to include a hysteresis zone around the center of the image so that a tracked face that is "close enough" to the center doesn't move the servos.

Step 6: Tuning the PID Controllers

This is perhaps the trickiest part of the project. You'll have to tune the PID controller and move_time values until you get desired results. There is plenty of good advice out there on tuning and debugging PID controllers, but values are very application-dependent, so you'll have to really understand what the values of the PID controllers do in order to get desirable results.