CompRobot - Computer Vision

Diana Vermilya and Nathan Lintz

November 2014

1 Goal

The goal of this project was to control a Neato robot using hand gestures. To achieve this goal, we developed a set of tools which detect a users hand position and map the position to a set of Neato commands. We developed a robust system to recognize different static hand gestures using openCV.

2 Methods

Our code detects user hand motions using openCV and controls the Neato using ROS. A user sits in front of their web camera can control the movement of a Neato using their hands.

2.1 Computer Vision

The vision system takes video inputs and performs computer vision operations to detect hand motions. The first stage of the vision module calibrates the camera. A user holds their hand up and sets calibration points which are then tracked in following frames. The frames then undergo contour detection and filtering which identifies the shapes which most likely resemble the users hand. The next step uses convex hulls and convexity defects to detect the users fingers.

Convex hulls are the smallest convex polygon which contains all points in the hand contour. Convexity defects are the points on the hand contour which are farthest from the convex hull. By combining the data from the convex hulls and convexity defects you can determine the number of fingers holding up as the defect points tend to be the point between two of a user's fingers.

Given the number of fingers and a hand contour, the algorithm decides whether the user is holding up a stop gesture or a directional gesture. Stop gestures are when the algorithm detects all five fingers are up while all other gestures need a direction to direct to the robot to.

To determine the robots direction, our algorithm finds the center of mass for the hand contour and which finger the user is using the point. It then finds the vector between the center of mass and the finger position to determine what direction the user is pointing. This direction is mapped to motor controls which can be defined arbitrarily by the user. For the sake of our demo we set the robot to move in the direction we were pointing.

2.2 Motor Controls

The motor controls from the Neato give the user the ability to move the robot from their computer. The Neato controller observes gesture events from from the Vision system and sends corresponding actions to the Neato's motors.

3 Design Decisions

An interesting design decision we made was structuring our code using reusable utility classes in our Helpers.py file. One such function is the renderer class. This object holds window objects which represent an openCV window and associated events. On each update call, the renderer draws the windows its holding and fires any event handlers that should be run. This class helped us immensely because it allowed us to register keypress events and it decoupled view logic from business logic. OpenCV doesn't support keypress events in their windows so this class made it easier to develop a user interface for our program because we could fire events on both mouse clicks or keypresses. Moreover, decoupling the rendering logic from the business logic made our code much cleaner since classes which dealt with processing images didn't have to be concerned with how the images are drawn for the user.

Another interesting design decision we made was when we tried to implement methods to evaluate a hand gesture. We tried using SIFT, SURF, and ORB recognition but found that none of them had a high accuracy rate. We instead decided to develop our own algorithm which involved looking for a hand's center of mass and finger position. This turned out to be a highly effective way to recognize gestures.

4 Code Structure

We structured the computer vision algorithm via Object Oriented Programming. Mirroring the algorithm as described above, we used the following structure: CallibrationController - obtains the relevant HSV data for background subtraction. BackgroundSubtractor - extracts a stream of black and white image frames TrackingController - extracts convex hulls and their associated defects GestureMatchService - used by TrackingController to choose the most likely gesture HandGestureRecognizer - Director class which instantiates the other classes and maintains the operation while it is not canceled.

5 Challenges

Parallelization of Tasks The structure of the problem we worked to solve was serial. For example, it was difficult to design and implement the convex hull algorithm without reliable background subtraction. It was also difficult to match defects to gestures before we had a good idea of how many defects would be found or they would be positioned. In retrospect, we would have benefited greatly from creating sample data generated by existing code, tutorials, or manual creation. Another strategy would have been to pair program or divide up tasks very rigorously such that each programmer had clear ownership over a section or set of sections.

Uncertainty in predicting applicability of known algorithms to our project. Over the course of this project, we partially or completely implemented many different algorithms, including k-means clustering, niave bayes, convex hulls, SIFT, intensity-based filtering, OpenCV's matchShape function, as well invented algorithms related to the specific gestures we were working with. These techniques ranged from being very effective to less effective than the null hypothesis. A lot of development was spent evaluating the effectiveness of algorithms that we did not end up incorporating into our final project. While this is an inherent risk in any research-style project, more initial research could have helped us to invest our time more effectively.

6 Project Improvements

In the next iteration of this project, we would like to implement motion based gestures, more complex Neato behavior, and improve our existing algorithms. We chose not to implement any motion gestures e.g. swiping left and right because we couldn't come up with compelling use cases for these gestures that wasn't already satisfied by pointing left or right. However, implementing these behaviors would involve using either the Lucas-Kanade method for motion tracking or developing our own algorithm for describing motion gestures. For example, we considered tracking the center of mass of the hand to determine its motion.

In terms of implementing more complex Neato behavior, we would have liked to tell the Neato to look for specified targets and follow them. We thought it would be a novel user experience to be able to point to an object in a room and be able to follow it. To implement this algorithm we would probably try to use SURF to match images from the users camera to the Neatos camera. We would then try to command the Neato to move to an object based on where it believes the user is pointing in the image.

In order to implement more gestures and complex Neato behaviors we would like to improve our existing algorithms. At the moment, we can only get extremely reliable behaviors when we wear a glove. It would be better if we could track human hands more accurately. To achieve this goal we need a better way of filtering out similarly colored objects such as the user's face.

7 Interesting Lessons

An interesting lesson we learned in this project was that its important to design your project plan with smaller modular functions in mind. Our algorithm design was overly monolithic which stemmed from the linear way we described our algorithm when we were planning this project. Instead, it would have been more useful to describe all of the components of the project individually and focused on smaller testable units which would combine to form more complicated behavriors. This way, we would have developed more code in parallel which would have resulted in a better overall project. Moreover, it would have made our code far more reusable in future projects.