Einstein TurtleBot

Scott Matsuo and Philip Arola

https://github.com/philiparola/ece478-turtlebot

# Overall Behavior

Our robot is designed to wave at the person closest to it. It does this through facial detection, and gestures according to the position of the person.

# Subsystems

## Gestures

Our robot utilizes ten servos with four in each arm and two in the head. In the following table, we defined the range/limits of the robot to prevent it from performing actions that could damage the servos. The configuration was done using the Roboplus Dynamixel Wizard, which allowed us to define parameters found in the table below. The first column identifies where the servo is located physically on the robot. The ID is the ID assigned to the servo which allows us to unique identify and command each servo. The Min (raw) and Max (raw) values are the minimum and maximum value that can be sent to the servo. This value can range from 0 to 1024 (0 to 300 degrees of rotation). It is important to limit this so the robot does not perform a movement where it can damage itself. The torque is the max percentage torque the servo can use multiplied by 10, i.e. 400 = 40%. This prevents the robot from swinging wildly and damaging itself as well. The Min (deg) and Max (deg) are the raw values translated into human readable degrees. The Init, Min, and Max (yaml) are the values used in the yaml file which determines the range of values you can publish to the servo. For the most part, we’ve made the init value equal to the min except in the case of the horizontal neck servo. By putting the init value exactly in the middle, when we publish the value 0 to the servo, the servo returns to the middle, causing the head to turn straight forward. The Min and Max (rad) values is the range of values you can publish to the servos. These values are in radians and publishing a value lowering than the min is equivalent to publishing the min, and the same goes for the max. Finally the last column describes most specifically what part the servo is controlling.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | ID | Min (raw) | Max (raw) | Torque | Min (deg) | Max (deg) | Range (deg) | Init (yaml) | Min (yaml) | Max (yaml) | Min (rad) | Max (rad) | Description |
| Right Arm | 1 | 409.20 | 818.40 | 400 | 120 | 240 | 120 | 409 | 409 | 819 | 0.00 | 2.52 | Shoulder Vertical |
| 2 | 341.00 | 647.90 | 400 | 100 | 190 | 90 | 341 | 341 | 648 | 0.00 | 1.89 | Shoulder Horizontal |
| 3 | 306.90 | 716.10 | 400 | 90 | 210 | 120 | 306 | 306 | 717 | 0.00 | 2.52 | Elbow Vertical |
| 4 | 341.00 | 647.90 | 400 | 100 | 190 | 90 | 341 | 341 | 648 | 0.00 | 1.89 | Elbow Horizontal |
| Left Arm | 5 | 204.60 | 613.80 | 400 | 60 | 180 | 120 | 204 | 204 | 614 | 0.00 | 2.52 | Shoulder Vertical |
| 6 | 341.00 | 647.90 | 400 | 100 | 190 | 90 | 341 | 341 | 648 | 0.00 | 1.89 | Shoulder Horizontal |
| 7 | 613.80 | 1023.00 | 400 | 180 | 300 | 120 | 613 | 613 | 1023 | 0.00 | 2.52 | Elbow Vertical |
| 8 | 409.20 | 682.00 | 400 | 120 | 200 | 80 | 409 | 409 | 682 | 0.00 | 1.68 | Elbow Horizontal |
| Head | 9 | 358.05 | 664.95 | 400 | 105 | 195 | 90 | 511 | 358 | 665 | -0.94 | 0.95 | Neck Horizontal |
| 10 | 375.10 | 613.80 | 400 | 110 | 180 | 70 | 375 | 375 | 614 | 0.00 | 1.47 | Neck Vertical |

Additionally, we defined poses with the values you publish in order for the robot to accomplish desired pose. Below are a few examples, where each ID corresponds to a servo.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Raise Right Arm | 2.52 | 0.95 | 1.05 | 1.05 | 2.52 | 1.89 | 1.05 | 0 | 0 | 0.8 |
| Raise Left Arm | 0 | 0 | 1.05 | 1.89 | 0 | 0.84 | 1.05 | 0 | 0 | 0.8 |

With all this information, we configured the robot to wave his arm and turn the corresponding direction the face is on.

## Vision

The robot utilizes any webcam compatible with OpenCV. The algorithm is fairly simple: our Python script utilizes the default Haar Cascade Classifier for face detection. From there, we calculate statistics that we need for our gesture decisions. We count the number of faces (the length of the array of face coordinates returned by the Haar Classifier), and the size of each face (by calculating the area).

In addition, we added a couple of elements to the display to assist/clarify the robot behavior. Specifically, we illustrate the faces that are detected by drawing a box around the face, with indicators for the center of the faces. We also draw the deadzone on screen. This is the zone where if a face is present, the robot will wave. Using fuzzy logic; if there is a face detected, but not within this deadzone, it will not gesture. Further, the vision module reports the position of the largest face relative to the center of the image. This allows us to determine which side the face is on, so we can change which side of the robot gestures. Calculating the face size is a sufficient proxy for distance calculation. This provides a very clearly defined behavior when there are multiple faces in the image.

# Assembling the robot, setup the development environment

There are three data connections to the controller; the TurtleBot base (the wheels), the servo controller (Dynamixel), and the Orbbec depth camera. These devices are hooked into a USB hub, which is then plugged into the main controller.

The controller utilizes Ubuntu 16.04, and uses ROS Kinetic.

# Lessons: what we learned/improvements

One improvement we would have made was to have the head turning mechanism follow the user based on the distance from the center rather than going all the way right or left. This would create an effect that the robot is tracking/watching you.

I would like to refine the gesture controls to be state-based. This would allow us to “stack” animations on to of each other, which would allow them to run multiple at once, with variable timing.

I would also like to refine the facial detection algorithm, because we go alot of false positives (eyes and mouths are recognized as faces within faces) and false negatives (mostly to do with the angle of a person’s head).

# Challenges

One challenge we frequently faced was intermittency due to wiring. Sometimes the robot would move and loosen a wire but not completely unplug it. We would then have to unplug and plug the wires in until we found the problematic connection. Eventually we replaced most of the wires completely and have found to have less intermittency issues. However, over time we would expect to see this come back.

We also encountered difficulty working on this project collaboratively, in that the catkin project files didn’t work too well across different development PCs. We had to utilize our version control very carefully to not break eachother’s projects.

# Video Link

<http://drive.google.com/file/d/184OHAJzbtnn5vT6z2YnNI3qG_jk8N05T/view>