Emotions and Face Actions

of

Baxter Research Robot

Supervisor:

Assistant Prof. Emre Uğur

Computer Engineering Department, Bogazici University

e-mail: emre.ugur@boun.edu.tr

Superviese:

Bilgehan NAL

Bachelor Degree of Computer Engineering Department at

Marmara University

e-mail: bilgehannal@marun.edu.tr / bilgehanl.03@gmail.com

Table of Contents

1. Introduction

Baxter is an industrial robot predecessor to Sawyer (robot) built by Rethink Robotics, a start-up company founded by Rodney Brooks. It was introduced in September 2012. Baxter is a 3-foot tall (without pedestal; 5'10" – 6'3" with pedestal), two-armed robot with a 1024 x 600 display. It weighs 165 lbs without the pedestal and 306 lbs with the pedestal. It is used for simple industrial jobs such as loading, unloading, sorting, and handling of materials. Brooks stated that Baxter was designed to perform the dull tasks on a production line. It is intended to be sold to small and medium-sized companies.

2. Background

Baxter has a screen which has 1024 * 600 display and it has sensors surrounding its head that allow it to sense people nearby. Other sensors around its head give the Baxter Research Robot the ability to adapt to its environment, unlike other industrial robots which will either continue to do their one task repeatedly, or will shut down and stop working at the slightest change in their environment. For example, Baxter is adaptive enough to know that it cannot continue with its job if it drops a tool, whereas some robots will simply continue to attempt to perform their job despite lacking the proper tools. Baxter runs on the open-source Robot Operating System on a regular, personal computer which is embedded in its chest. Baxter can be placed on a four-legged pedestal with wheels to become mobile. Baxter also has extra sensors in its hands that allow it to pay very close attention to detail.

2.1. Technologies Used for the Project

All the codes used in the project are written in the python programming language and the image files used for the Baxter's face are created using Adobe Photoshop CC.

2.2. Screen of Baxter

The Baxter Robot head display is a 1024 x 600 SVGA LCD screen, and is published to through the "/robot/xdisplay" topic.

2.2.1. Display Image on the Screen

The user can display the ROS Images on the screen of Baxter Robot. Therefore publisher node needs some libraries to display the image such as “cv2, cv_bridge, sensor_msgs.msg.Image”

It is important to read the images as 2D numpy arrays in OpenCV 2.0 in order to execute various OpenCV algorithms on them. The imread() function is an OpenCV function that can be used to load images as 2D numpy arrays from given path.

The OpenCV image has to be converted into ROS image message in order to send them as ROS messages or perform any ROS functions on it. The cv_bridge library is used to convert between ROS and OpenCV images. cv2_to_imgmsg function in cv_bridge library converts the OpenCV image to ROS image

The ROS image can be published to the /robot/xdisplay topic with message type Image from sensor_msgs.msg

Example Code:

```
img = cv2.imread(path)  # Reading image from given path
msg = cv_bridge.CvBridge().cv2_to_imgmsg(img, encoding="bgr8") # Converting image to ROS Image
# Publishing image
pub = rospy.Publisher('/robot/xdisplay', Image, latch=True)
pub.publish(msg)
```

2.3. Head Sonar Sensors

Around the head is a ring of 12 sonar sensors for detecting movements within Baxter's proximity. Next to each sonar sensor are yellow LEDs, which by default indicate when the corresponding sensors detect an object within a certain threshold.

Mounted in a ring around the head are 12 sonar distance sensors. There is also a yellow LED around each sensor that, by default, triggers when the corresponding sensor measures a distance below its threshold.

Component IDs: head_sonar

2.3.1. Usage of Head Sonar Sensors

The sonar range measurements are published as a set of 3D coordinate points in space around Baxter, called a PointCloud. This gives a "mapping" of detections in the workspace and is well-suited for occupancy-oriented reasoning in the robot's world.

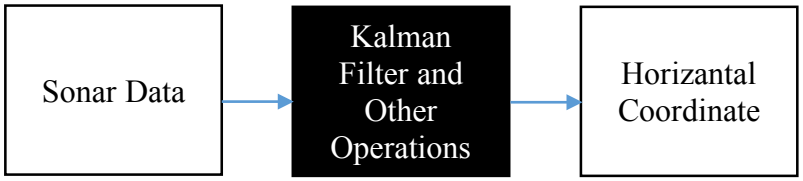
Topic for reading sonars 3d distances: /robot/sonar/head_sonar/state

```
rospy.Subscriber('/robot/sonar/head_sonar/state', sensor_msgs.msg.PointCloud, callback)
```

As we can see from the above code, the message type is PointCloud from sensor_msgs.msg. This message includes the information about the sensors which determines an object. Information includes x, y, z coordinates and distances of the sonar.

2.3.2. Object Following

Eye of the Baxter can move in two dimensions (Vertical and Horizontal), when we turn on the object following mode (human following mode), Baxter follows the object (human) just horizontally. We cannot detect the z coordinate of the human's head from the data coming from sonars. Because of this reason, we have to fix vertical position of the eye to coordinate of 0. Dataset coming from the sonars of Baxter are noisy. We have to eliminate this noise to use the data for following object.
There are some operations and "Kalman Filter" used for eliminating the data coming from sonars converting the noisy data to a horizontal coordinate.



Operations:

We took the dataset as a discrete number (id of the sonars). Difference of the value of two consecutive sonar is 1.
After the tests applicated on fixed body, we calculated the standard deviation of the data coming from the sonars.
Standard Deviation: 0.50635561

Current data is the data coming from the sonars momentarily. There are some unwanted values (wrong data) in current data. Elimination is required to use current data. We can take the distance value of a sonar so, we can see the unexpected values looking at the distances. If a sonar's distance value is much bigger than the smallest one then, we can say that this data was unwanted value.
If the current data is bigger than <percentage rate>%, we can ignore that value.
Determined value of percentage rate is 30.
After the elimination of the unexpected values, we have to ignore outliers too. Remaning data can be used in Kalman filter and other operations.

To use Kalman Filter, we need a reference value except the current data. Therefore we choosed the mean of last 35 value as a reference value.

After applicating the Kalman filter, we take a value between -3 and 3 [-3, 3]. To take better solution, we combine the result coming from Kalman filter and reference value. (97% Result coming from Kalman filter, 3% reference).

Lastly, we have to extend the value to value between -80 and 80 [-80, 80] (Horizontal coordinate of the Baxter's eye.)

2.4. Position of Hands

Cartesian endpoints are published at 100 Hz, the endpoint state topic provides the current Cartesian Position, Velocity and Effort at the endpoint for either limb.

- All of Pose, Twist, and Wrench measurements are reported with respect to the /base frame of the robot
- The endpoint state message provides the current position/orientation pose, linear/angular velocity, and force/torque effort of the robot end-effector at 100 Hz. Pose is in Meters, Velocity in m/s, Effort in Nm.

2.4.1. Usage of Position Information of Hands

Endpoint states are published as a set of 3D coordinate points in space around Baxter, called a EndpointState. This gives the endpoint state of a limb as Cartesian coordinate.

Topic for reading Cartesian coordinate of the limb: /robot/limb/<side>/endpoint_state

Example Subscribing:

```
rospy.Subscriber('/robot/limb/left/endpoint_state', EndpointState, callback)
```

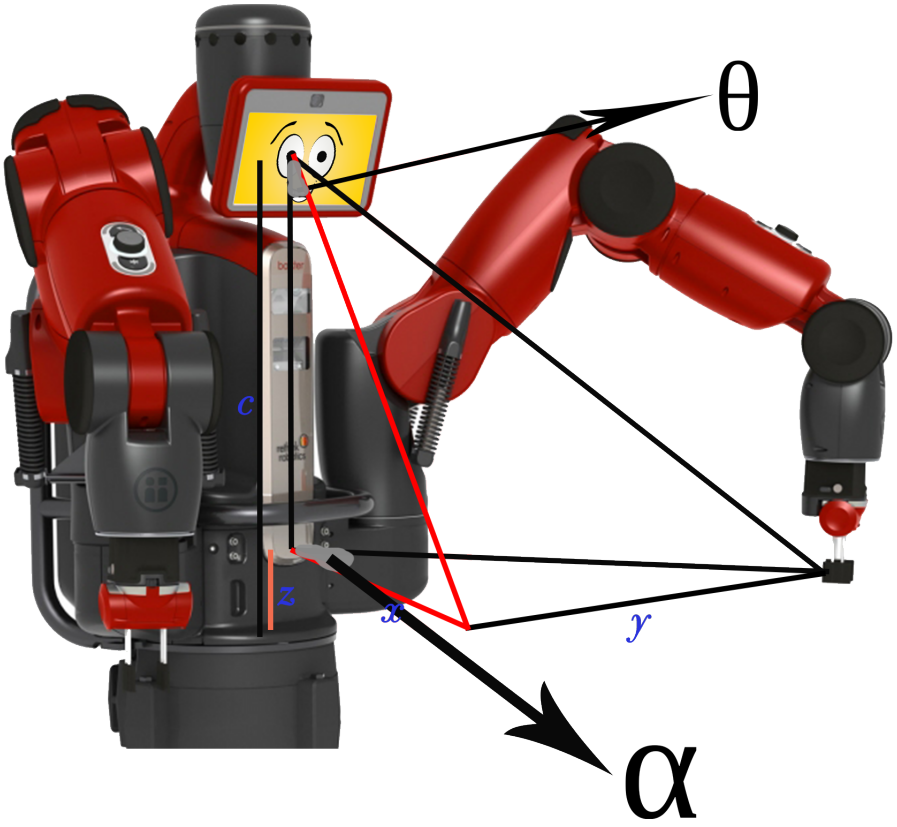
As we can see from the above code, the message type is EndpointState from baxter_core_msgs.msg.
This message includes the x,y, z coordinates of a limb. Unit of the distances are given as unit of meter.

Information includes x, y, z coordinates and distances of the sonar.

Example code of reaching the x,y and z coordinate information from the message.

```
    x = msg.pose.position.x
    y = msg.pose.position.y
    z = msg.pose.position.z
```

2.4.2. Hand Following



Eye of the Baxter can move in two dimensions (Vertical, Horizontal) As we can see from the above image, we have two important angles to follow the endpoint of limbs. (θ, α)

If we scale the θ angle to $\frac{\pi}{2}$ − [-120, 120] and α angle to [-80, 80], we can obtain the position of where the robot should look.

Calculation of the α and θ

$\alpha = tan^{-1}(\frac{y}{x})$

$\theta = tan^{-1}(\frac{x}{c-z})$

Endpoint State data gives us the coordinates accepting the origin of Baxter's waist, but, in this we need to suppose the origin of Baxter's face to calculate the angles. Therefore "c-z" gives us the z coordinate of the abstract origin.

Exact coordinates calculation:

Vertical: α * 57.294

Horizontal: ($\frac{\pi}{2}$ - θ) * (-76.394)

3. Structure of the Project

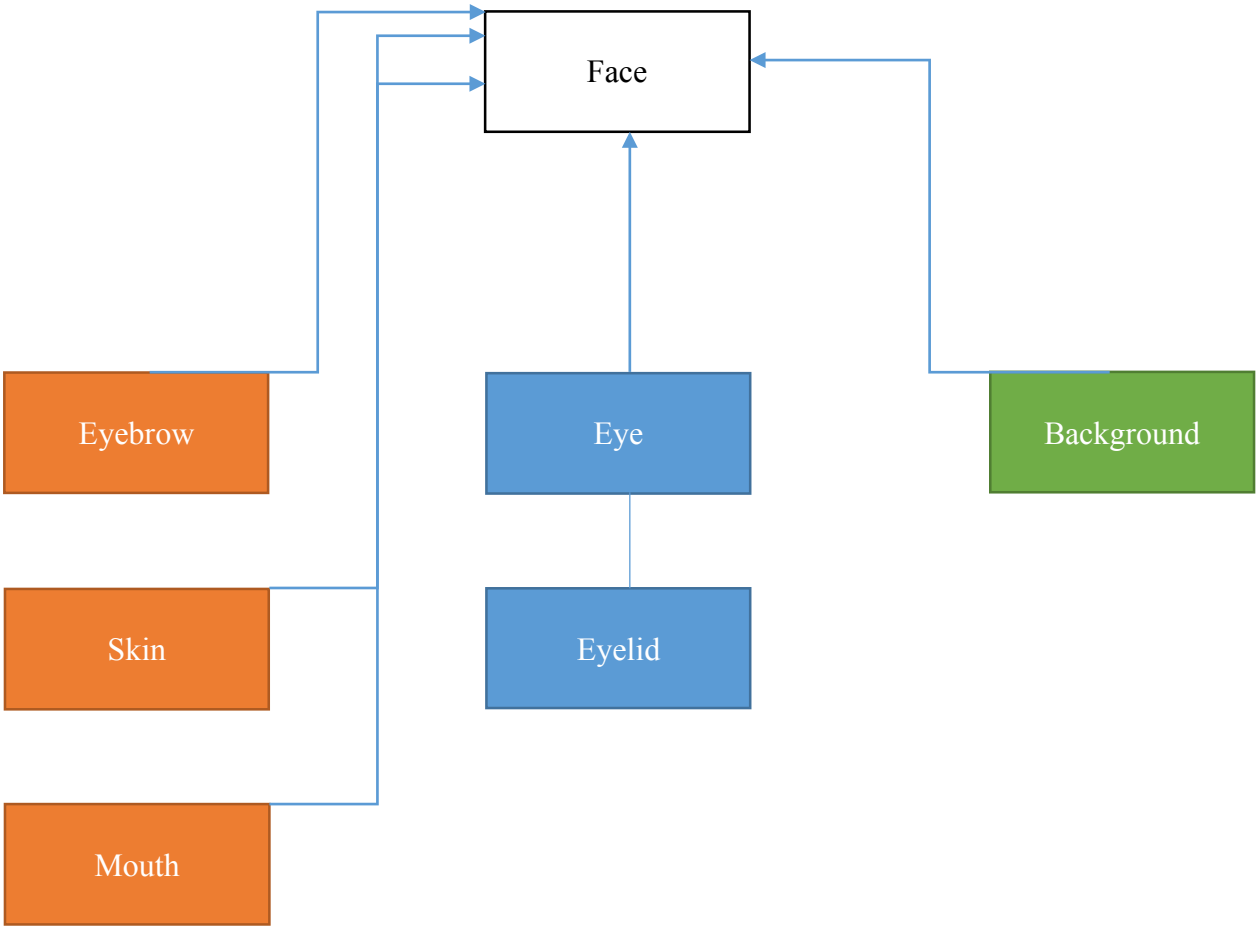
This project has mainly one node. This node subscribes two different topics. One of the subscriber is used for listening to information of sonar sensors. The other one subscribes a String topic to listen commands to behavior of the face of Baxter Robot (Emotions and Actions). Also the main project has a Face library, this library includes all face states and emotions.

The main node is consist of the threads. One of the threads is used for subscribing the topics. Second thread is used for the ordinary issues such as blink, following the object

We can show only pictures on the Baxter screen. Therefore when we applicates the actions (moving eyes, moving eyelids), the program regenerates the face picture and republishes.

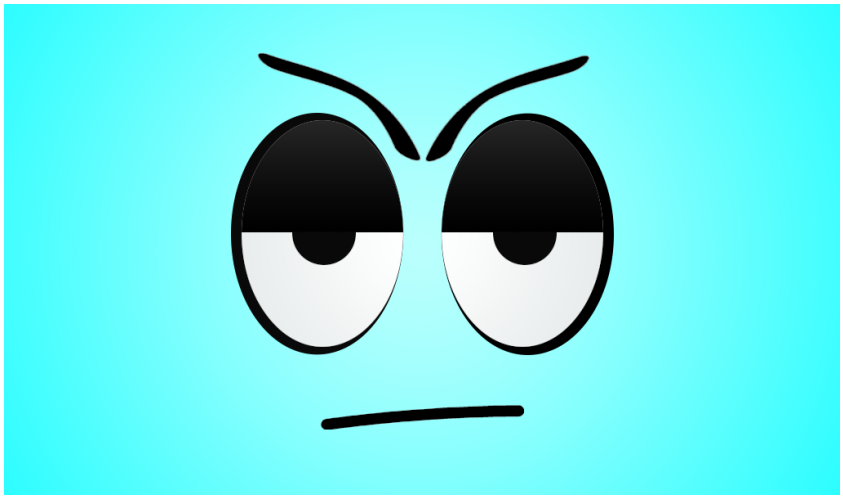
3.1. Modelling Face

The “Face” script can be used as a library from any node. The Face is consist of 5 main parts (Background, Skin, Eye, Eyelid, Mouth, Eyebrow) and “Face.py” is a script which combines the all face parts to each other.



Every part of the face has 1024 * 600 png pictures. These pictures could be considered as a layer.

You can see the generated face example from the face layers below.



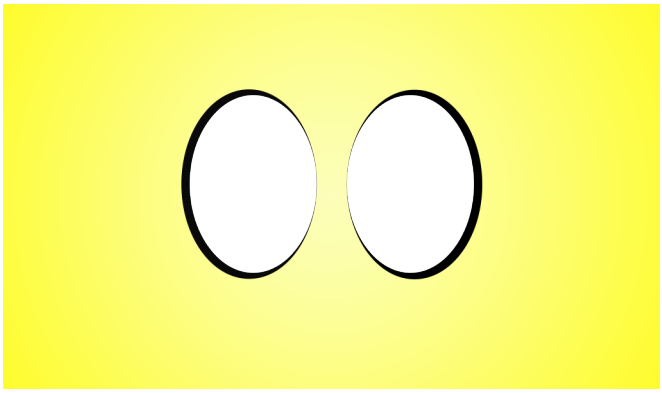
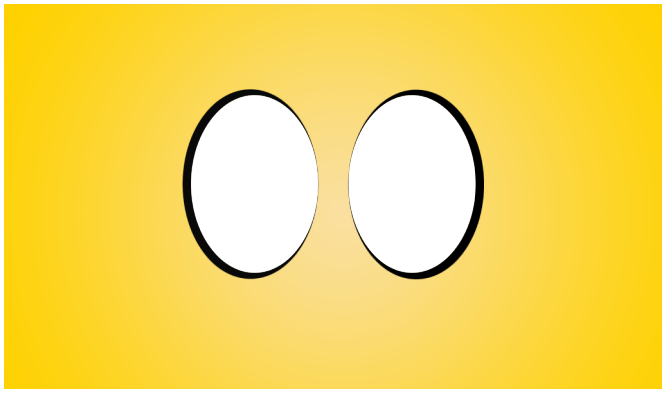
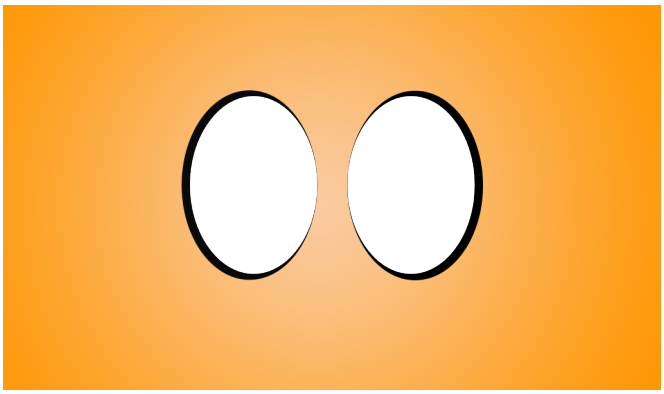
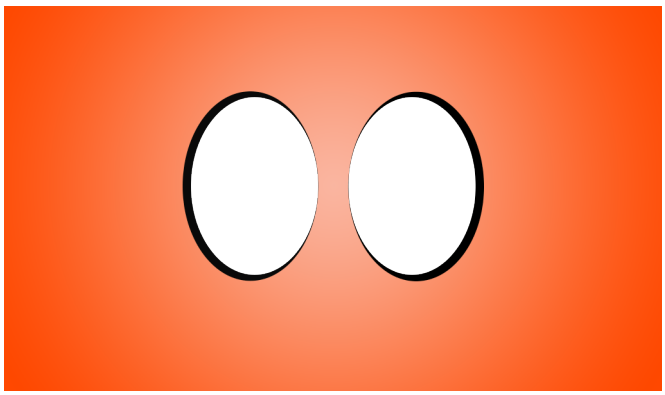
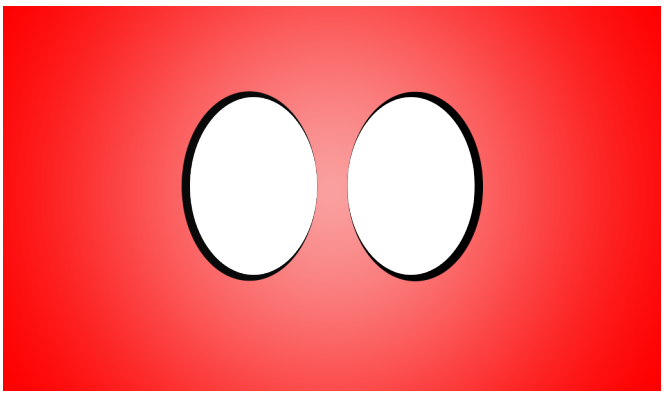
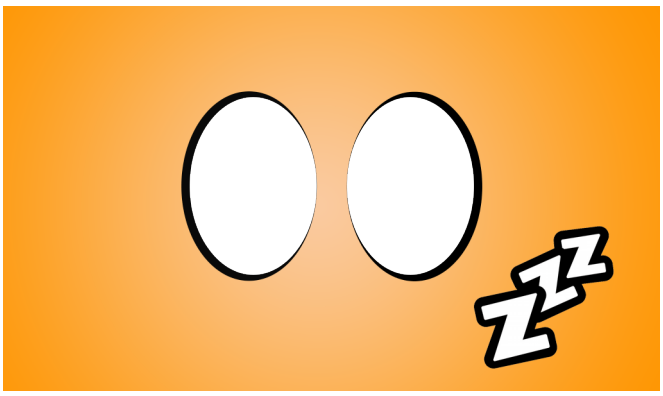
The parts of the face has different properties. Some of them is static object., Some of them is dynamic object. Only background is not a object. Background is a png file, and this image is used for every face types.

3.1.1. Static Objects

Static objects of the face are skin, eyebrow and mouth. These objects have more than one shapes, but nothing is moving.
Static objects has one constructer, there are definitions of the all versions of the part in an array inside of the constructers. Also it has current version of part of the face. We can change the current picture using encapsulation functions. (Setter function).

3.1.1.1. Skin

There are six types of skin. All of the skins have same shape except sleepy skin, but different colors. (Yellow to Red)

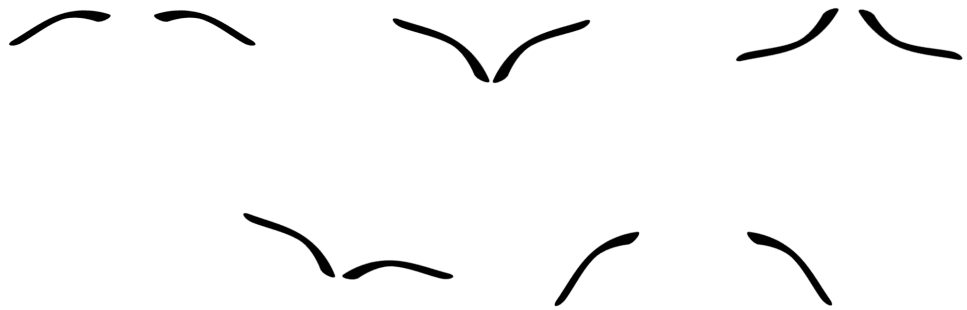







All skins are kept in an PIL.Image array.

When we run the program we can see the initial skin color was yellow.

3.1.1.2. Eyebrow

There are five types of eyebrow. All eyebrow image has 1024 * 600 resolution and there is a transparent background under the eyebrows. All eyebrows has different shapes.



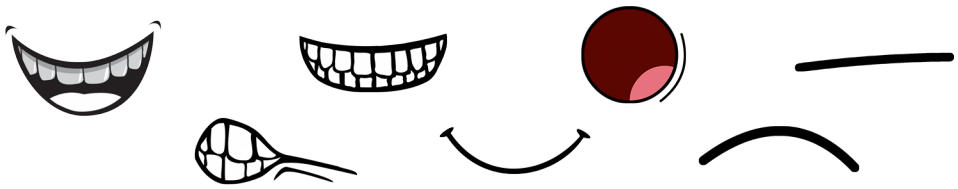
All types of eyebrows is created for the place of eye, when we merge skin and eyebrow. Location of the eyebrows on the skin will be perfectly.

3.1.1.3. Mouth

There are seven types of mouth. These are:

- Angry Mouth
- Bored Mouth
- Confused Mouh
- Sad Mouth
- Smiling open mouth
- Showing tooth mouth.
- Normal smiling mouth

Initial mouth shape is normal smiling mouth. The thing that determines the emotions mainly is



mouth. Combination of eyebrows and mouth is determined the current emotion.

3.1.2. Dynamic Objects

Dynamic objects of the skin are eyes and eyelids. These parts of the face have one type of shape but they have property to move. A dynamic object has variable(s) which keeps the position(coordinate) value as a pixel.

Eyes

All movements (actions) are changing of the coordinates in given time. If the programmer gives the exact coordinates without animation, but if the programmer wants animation s/he gives the time value then animation will be done in given time. For that reason program will set the its own fps rate.

3.1.2.1. Eye

Eyes are created with two filled circles in one image. When the programmer wants to move robot's eyes. S/he can make eyes move together.

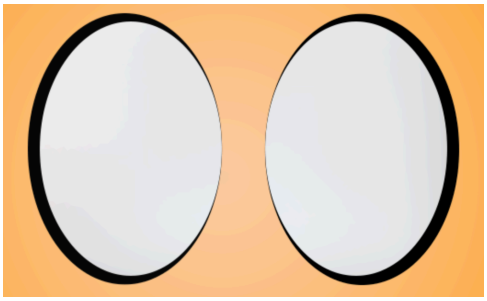
There are two types of function in the eye class. These are about movement of eye and calculation of animations.

```
lookExactCoordinate(self, x, y):
```

We can set where the robot is looking at with lookExactCoordinate function without animation.

```
lookWithMotionCalculation(self, x, y, destinationX, destinationY, totalTime, instantTime):
```

This function calculates the position of the eye in the instant frame, we will use the function in the Face Class for the movement of eye with animation.

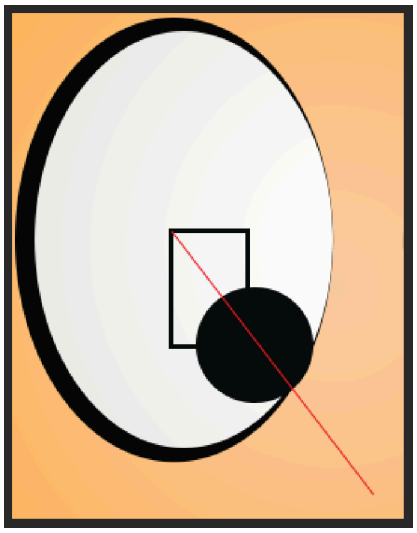


The eyes are moving in an elliptic place. Therefore, we have to determine the limits of the eyes for its movement.

Properties of the elliptic place:

Vertical radius: 120 px
Horizantal radius: 80 px

We can take the position of the eye and we can calculate the looking angle



When we look at the slope of the red line, we can reach the information about the angle using trigonometric functions.

If the eyes are at outside of the ellipse. We have to reposition the eye without changing the looking angle. (Saving the angle scale the position according to the radius of the ellipse given angle.) Ellipses have different radius in every different positions. If we know the angle we can reach the radius with below formula.

$$r = \frac{80 * 120}{\sqrt{a^2 sin^2\theta + b^2 cos^2\theta}}$$

3.1.2.2. Eyelid

The eyelid has same animation skills with the eye, but it can move only vertically. It was created with and gradient picture.

Default position of the eyelid is eye closed position. When we want to open the eyes, we have to set the position at the -330 px.

3.1.3. Face Functions

```
buildFace(self):
```

buildFace function is used for combining all face parts together. This function returns an image form of the face as a numpy array.

```
show(self, publish):
```

show function is used for showing the current image of face on the screen of the Baxter.

```
lookWithMotion(self, cv2, destinationX, destinationY, time, publish):
```

we can use lookWithMotion function when we want to make move the robot's eyes with an animation. The eyes will go to the destination position if the destination positions in the ellipse of eye in given time.

```
wink(self, cv2, publish)
```

wink function provides with the blinking motion of the Baxter.

In addition to the these functions, there are some emotion functions like below.

```
emotion_default(self, cv2, publish):
emotion_happy(self, cv2, publish):
emotion_angry(self, cv2, publish):
emotion_confused(self, cv2, publish):
emotion_sad(self, cv2, publish):
emotion_panic(self, cv2, publish):
emotion_bored(self, cv2, publish):
emotion_crafty(self, cv2, publish):
sleep(self, cv2, publish):
wake_up(self, cv2, publish):
```

3.2. Subscriber

The main node of the project is “screen_listener.py”. This script subscribes two topics and makes some ordinary issues. There are two threads in this script. One of them is used for the ordinary issues such as blinking and object follow. Other one is subscribing the topics.

Two diffirent toppics are subscribed. These are “/robot/sonar/head_sonar/state” and “display_chatter”. Display chatter topic is used for the commanding the program. (Looking, Emotions etc.) “/robot/sonar/head_sonar/state” is used for taking sonars’ informations when we command “object follow”

Also this script has a function named publish_image

```
publish_image(img):
```

This function is used for the publishing given image on the Baxter’s screen.

4. Import and Usage

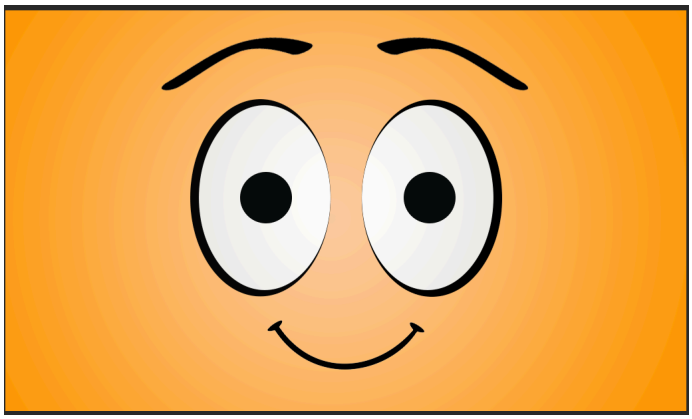
This project has more than one script and image files because of this reason, we decided to make a package named “baxter_face”. User have to import package named “baxter_face”.

4.1. Import the Project

The programmer should applicate below steps to run the project on the Baxter Robot

- Copy the ROS package of “baxter_face” in a src folder in work space.
- Applicate the command of “catkin_make”
- When we want to run the program we can use this command in the terminal screen.
  rosrun baxter_face screen_listener.py
- Anoyher way to start the program is roslaunch. if the user uses the commands below, Baxter Face Program and button controller will be enabled.
  roslaunch baxter_face baxter_face.launch

After applicating the all steps. Before the program runs completely, the program tries all images on the Baxter screen We can see the default face on the Baxter. (Default face can be looked like blue because of the encoding format)



4.2.Using

After run the “screen_listener.py”, the terminal screen shows you some informations about the robot motion. We have to give some commands to control the face of Baxter Robot.

Commands are sending to “screen_listener.py” as a message type String, with topic of “display_chatter”.

Example terminal command:
rostopic pub /display_chatter std_msgs/string <command>

Also the programmer can send the message from a publisher node.

4.2.1. Emotions

There are eight types of emotions. These are:

- Default Emotion
- Happy
- Sad

- Angry
- Confused
- Panic
- Bored
- Crafty

Some emotions are occurred with more than one combination of the mouth and eyebrows. When the programmer choosed the emotion if that emotion has many combinations. The program will chose one of them randomly.

Control the emotions with terminal screen:

- Default Emotion : rostopic pub /display_chatter std_msgs/String default
- Happy : rostopic pub /display_chatter std_msgs/String happy
- Sad : rostopic pub /display_chatter std_msgs/String sad
- Angry : rostopic pub /display_chatter std_msgs/String angry
- Confused : rostopic pub /display_chatter std_msgs/String confused
- Panic : rostopic pub /display_chatter std_msgs/String panic
- Bored : rostopic pub /display_chatter std_msgs/String bored
- Crafty : rostopic pub /display_chatter std_msgs/String crafty
- Wake up : rostopic pub /display_chatter std_msgs/String wake_up
- Sleep : rostopic pub /display_chatter std_msgs/String sleep

4.2.2. Actions

We can use the same type of message to see animations.

Control the animations with terminal screen:

Looking somewhere:

rostopic pub /display_chatter std_msgs/String look_<x coordinate>_<y coordinate>
rostopic pub /display_chatter std_msgs/String look_<x coordinate>_<y coordinate>_<time>
rostopic pub /display_chatter std_msgs/String dynamic_look_<x coordinate>_<y coordinate>

rostopic pub /display_chatter std_msgs/String dynamic_look_<x coordinate>_<y coordinate>_<time>

There are two main commands for eye movement. When we applicate the first command, eyes are moving towards to given coordinates in the 0.5 seconds. However, we can use second command if we want to give animation time manually.
If we could use dynamic keyword before the command, robot will look at the given coordinate with the combination of head movement and eye movement.

Wobbling and Robot Situation Control:

rostopic pub /display_chatter std_msgs/String wobble_<angle>
rostopic pub /display_chatter std_msgs/String enable
rostopic pub /display_chatter std_msgs/String disable

There are two commands for controlling the robot situation (enable - disable), user can enable or disable the Baxter robot publishing a String message on display_chatter topic.
If the robot is in enable situation, user can change the position of tha head giving an angle as a radian.

4.2.3. Object Follow

We have to pulishb a message to “display_chatter” topic to follow the object with the eyes of the Baxter. Terminal version of the command is:

rostopic pub /display_chatter std_msgs/String human_follow_on
rostopic pub /display_chatter std_msgs/String human_follow_off

if we send the “human message” with on command, Baxter is following the object(human) with its eyes. To cancel the following, we can send “off” command with an “human” command.

4.2.4. Arm Follow

We have to publish a message to “display_chatter” topic to follow the arm with the eyes and eyes with head.
Terminal version of the command is:

rostopic pub /display_chatter std_msgs/String left_arm_follow_on
rostopic pub /display_chatter std_msgs/String right_arm_follow_on
rostopic pub /display_chatter std_msgs/String dynamic_left_arm_follow_on
rostopic pub /display_chatter std_msgs/String dynamic_right_arm_follow_on

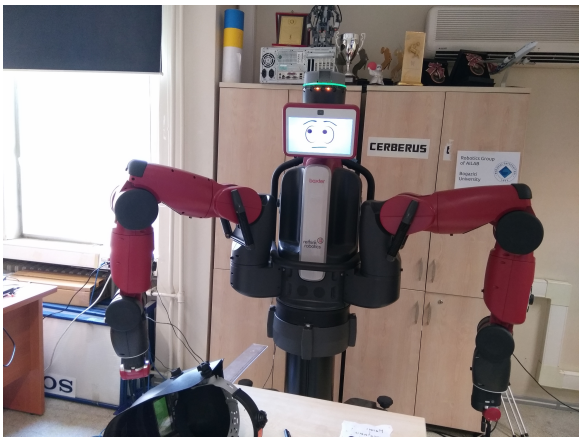
We can provide with the robot can follow its arms with eyes or eyes and head movment.

4.2.5. Exit

If the programmer closed the “screen_lister.py” clearly, s/he can send “exit message”
Example: rostopic pub /display_chatter std_msgs/String exit

5. Result

In this project, We tried to display a cartoon face on the screen of Baxter Robot using ROS and Python. When I run the program in the simulation, there was no problem but in the real robot. The colour of the skin was blue. Reason of this difference was about the encoding of the image.



There is a few problem when the program was running on the real robot for example when we use wireless connection animations’ frame per second rate was very low, but if we have network connection with a wire, we can see satisfying results. Also user can connect the Baxter’s internal PC using ssh connection and run the program in Baxter’s internal pc (Recommended way). User can take the best result using the internal pc.

6. Source Materials

- http://sdk.rethinkrobotics.com/wiki/Head
- http://sdk.rethinkrobotics.com/wiki/API_Reference#screen-xdisplay
- http://sdk.rethinkrobotics.com/wiki/Screen
- http://sdk.rethinkrobotics.com/wiki/Sonar_Control
- https://en.wikipedia.org/wiki/Robot_Operating_System
- http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages
- http://wiki.ros.org/cv_bridge
- https://en.wikipedia.org/wiki/Robot
- http://www.effbot.org/imagingbook/image.htm
- http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_image_display/py_image_display.html
- https://en.wikipedia.org/wiki/Ellipse
- http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29