Visualization in ROSproessingjs

Sungmin Lee

Brown University
Department of Computer Science
Visualization of Robot's side

- One of fundamental ideas of Processing is easy visualization.
  - Of course using internal processing functions is still easy, but..
- How do we visualize what robot sees on our client machine (browser)?
  - Remember that it is available subscribing everything as long as it is currently publishing.
  - Subscribing is the key!
Subscribing GStreamer message

- If you once publish GStreamer message from a robot, you can subscribe it on your client.
- In fact, the publish message is NOT necessarily to be GStreamer as long as the type of publish message is *rgb*.
  - Fortunately, “sensor_msg::Image” uses rgb data type.
- Another important point is that you should access `<msg_name>.uri`, not just `<msg_name>`.
  - Examples are ready.
Subscribing GStreamer message

- Here is an abstract structure how it works.
- As you see, you could freely communicate with the server by using `publish()` and `subscribe()` functions.
Sample Code: subscribe GStreamer

- First of all, you should subscribe the message.

```c
subscribe('/gscam/image_raw', getCamStream);
```

- This function means your program will repeatedly call `getCamStream()` function which has '/gscam/image_raw' as an argument.

- Thus, your `getCamStream(msg)` would look similar to this:

```c
void getCamStream(msg)
{
    if(lock2)
        return;

    lock2 = true;
    img = loadImage(msg.uri);
    lock2 = false;
}
```

- Where `loadImager(rgbdata)` is an internal processing function which converts rgb data to Pimage data type.
Example #1: Object Seeking (1/2)

- In the same way, you can also get blob information from the robot side by subscribing '/blobs' message.

```java
subscribe('/blobs', getBlob);
```

- This subscribe function will call `getBlob()` function repeatedly, and `getBlob()` function should pass the message to local variables.

```java
void getBlob(msg)
{
    if(lock1 || msg.blob_count == 0)
        return;

    lock1 = true;
    for(int i=0; i<msg.blob_count; i++)
    {
        //get blobs which have the same color as target only
        blobList.add(new CBlobInfo(msg.blobs[i].red, msg.blobs[i].green));
    }
}
```

- Where `blobList` is an `ArrayList` of `CblobInfo` class which contains every single blob datum of each frame.
Example #1: Object Seeking (2/2)

- The full code and detail explanation of object seeking is available on Brown wiki page.

- Also, Youtube video clip is also available here:

  http://www.youtube.com/watch?v=ZyQ96GDJft4&feature=player_embedded
Example #2: Object Tracking (1/2)

- You can also publish your movement by calling `move_robot(x, z)` function which is a wrapping function of 'geometry_msgs/Twist' publisher.

- By doing that, you can interactively move your robot based on the location of blobs.

```c
//move toward to the targetBlob
void trackBlob()
{
    //if you don't see the targetBlob or you are close enough, Stop
    if(targetBlob.size == 0 || targetBlob.size > 150000)
        move_robot(0.0, 0.0);
    else
    {
        int blobCenter = targetBlob.left + targetBlob.width/2;
        int scrCenter = scrWidth/2;

        float kp = 0.01;

        //Set rotation speed proportionally according to the blob's position.
        int diffX = blobCenter - scrCenter;
        float zVal = kp * diffX;
        move_robot(0.1, -1 * zVal * 0.1);
    }
}
```

- Where `targetBlob` is a class which contains the biggest blob data so that robot can track it.
Example #2: Object Tracking (2/2)

- The full code and detail explanation of object tracking is available on Brown wiki page.

- Also, Youtube video clip is also available here:
  http://www.youtube.com/watch?v=8IzGQXdKblE&feature=player_embedded
Limitation and extension

- One of the biggest limitations of Processingjs is that you cannot use java libraries such as openCV, openGL since it is a pure javascript.

- However, it also means that you may use all the internal functions of Processingjs without limitation to display things on your browser very simply.

- It is encouraged to make an importable ROS processing library(java) so that we could use all the java libraries for ROS visualization.