APÉNDICE

A

PYROBOTICS

A.1. Detalles del uso del BlackBoard

BlackBoard es una herramienta de paso de mensajes y respositorio de variables compartidas (más o menos equivalente a ROS: `http://ros.org`) desarrollada en el laboratorio de Bio-Robótica en una tesis de maestría [5]. Sirve para comunicar diferentes programas bajo los esquemas comando-respuesta y publicación-subscripción.

La manera en que está implementado es a través del esquema cliente-servidor, donde el BlackBoard toma el rol de cliente, y cada módulo que se quiere conectar, se comporta como servidor. El BlackBoard utiliza un archivo de configuración para conocer la ubicación de cada módulo (dirección IP y puerto) para establecer una conexión y poder comunicarse con ellos. Cada módulo puede entonces enviar mensajes que corresponden a comandos, que deberán ser ejecutados por otro módulo, o bien, por el mismo BlackBoard, en caso de ser comandos estándar que él entienda, por ejemplo el comando **write_var**, que sirve para escribir un valor a una variable compartida.

Los mensajes se envían en forma de texto y tienen el siguiente formato básico:

```
<nombre_comando> "<parametros>" @<id>
```

Donde el `<id>` es un número entero y sirve para identificar cada mensaje con su respuesta, `<nombre_comando>` puede ser cualquier cualquier cadena que pueda ser el nombre de una variable en el lenguaje C, y `<parametros>` es cualquier cadena de texto que el módulo destinatario tendría que interpretar.

Las respuestas incluyen adicionalmente, antes del id, un 1 en caso de que el comando haya sido exitoso, o un 0 en caso de que haya fallado, y pueden sustituir los parámetros por alguna respuesta que el comando deba generar, por ejemplo: una lista de objetos que el módulo de visión ha reconocido.

El esquema de comando-respuesta es adecuado para procesos que son ejecutados sobre demanda, es decir, en el momento en que se necesitan, se solicitan. Esto puede ser por varias razones, como que el robot necesite estar ubicado en alguna situación en particular, por ejemplo, un comando que sirva para abrir una puerta debería ser ejecutado solamente cuando el robot se encuentre frente a una puerta cerrada; o si el costo de procesamiento es muy alto para ejecutarlo continuamente, como los procesos de visión.

Sin embargo, muchas veces es conveniente reaccionar cuando algo en el ambiente sucede, independientemente del plan que se esté ejecutando, por ejemplo, cuando el robot escucha algún comando de voz. Para este tipo de problemas existe la comunicación de publicación-subscripción, en que cualquier módulo, que usualmente está sensando algo sobre el ambiente, puede escribir a una variable compartida, y todos los módulos que estén subscritos a ella recibirán una notificación de que el valor ha cambiado, para que puedan actuar en consecuencia si fuera necesario.

Para facilitar la implementación de nuevos módulos, se han desarrollado APIs de desarrollo que abstraen todos los detalles de conexión y manejo de mensajes de la aplicación para la que fue diseñada el módulo. Existen APIs para los lenguajes C#, C++ y python; en este apéndice se comenta un poco sobre el diseño de la API de python, llamada pyRobotics, y se muestran algunos ejemplos.

A.2. Documentación y ejemplos de pyRobotics

pyRobotics **no** está basada en las APIs de C# ni de C++, por lo tanto, su uso es diferente.

Un módulo desarrollado con la pyRobotics para conectarse con el BlackBoard necesita de dos partes básicas: [1]

1. Una rutina de inicialización donde:

   a) Proporciona la configuración necesaria para conectarse al BlackBoard y responder a los comandos que le corresponden.

   b) Deja todo listo para empezar a procesar comandos, dependiendo del módulo del que se trate.

   c) Inicia la comunicación con el BlackBoard.

   d) Opcionalmente crea y se subscribe a cualquier variable compartida que vaya a utilizar.

   e) Le avisa al BlackBoard que este módulo está listo para recibir comandos.

2. Un ciclo principal para hacer lo que sea que necesite hacer. Si el módulo sólo debería responder a comandos, se puede usar la función **BB.Wait()** para mantener un ciclo ocioso para prevenir que el programa termine (y por lo tanto, el ConnectionManager y el CommandManager, que se encargan de manejar la conexión y la recepción de comandos).

La rutina de inicialización debe incluir llamadas a las funciones **BB.Initialize**, **BB.Start** y **BB.SetReady**. *BB.Initialize* recibe básicamente un número de puerto y un mapa de funciones que define qué función ejecutará qué comando, *BB.Start* no recibe parámetros y sirve para inicializar la conexión con el BlackBoard; y *BB.SetReady* puede recibir un valor `booleano` (por default es `True`) indicando que el módulo está listo para recibir y procesar comandos[2].

[1] Realmente es el BB quien se conecta al módulo, pero como pyRobotics abstrae esto de la implementación, desde el punto de vista del desarrollador es más sencillo considerarlo como que los módulos se conectan a él.

[2] Esta función se puede llamar en cualquier momento para avisar al BlackBoard que su estado `ready` ha cambiado.

Los comandos más comunes, además de los arriba mencionados, que se pueden utilizar son los siguientes:

BB.Send Envía un comando SIN esperar por una respuesta. En general este comando es para uso interno, pero se provee la función en la API para casos en que realmente no importe la respuesta, o se trate de un módulo que contemple ejecución asíncrona de comandos y tenga otra forma de manejar las respuestas.

SendAndWait Envía un comando y espera por la respuesta para continuar la ejecución.

ReadSharedVar Es una función para consultar el valor de una variable compartida, sin estar subscrito ni tener que esperar por una notificación.

CreateSharedVar Crea una variable compartida a la que se publicará información.

WriteSharedVar Escribe (publica) un valor en una variable compartida.

SubscribeToSharedVar Se suscribe a una variable compartida. Cuando la variable cambie su valor (alguien más escriba en ella), éste módulo recibirá una notificación.

La figura A-1 muestra un ejemplo de un módulo sencillo que responde al comando `tst_testfunction`, y envía dos comandos, uno con la función `BB.SendAndWait`, y otro con la clase `ParallelSender`, para después quedarse esperando en un ciclo ocioso para continuar recibiendo comandos.

```
import time
from pyrobotics import BB
from pyrobotics.parallel_senders import ParallelSender
from pyrobotics.messages import Command, Response

# These command handler functions could (and probably should) be defined in another module
# as to keep the main program file simple and clean.
def testFunction(c):
  print "testFunction called..."
  print 'Params: ' + c.params
  return Response.FromCommandObject(c, True, 'Command response')

fmap = {
     'tst_testfunction' : testFunction
}


def main():
  BB.Initialize(2000, fmap)
  BB.Start()
  BB.SetReady(True)

  print 'Sending command say...'
  print BB.SendAndWait(Command('spg_say', 'This is a test.'), 5000, 3)

  print 'Sending Async command...'
  ps = ParallelSender(Command('othertst_slowfunction', 'This is another test.'), 5000, 3)

  while ps.sending:
    print 'sending...'
    time.sleep(0.3)

  print 'Response received...'
  print ps.response

  BB.Wait()

if __name__ == "__main__":
  main()
```

Figura A-1: Módulo sencillo desarrollado en python con el uso de pyRobotics.

Este ejemplo fue tomado de la documentación y está incluido con el código fuente de la librería. Para consultar el código fuente y otros ejemplos puede acceder a la página de GitHub del proyecto: `http://www.github.com/BioRoboticsUNAM/pyRobotics/`.

La liga a una documentación un poco más extensa está incluida en el README del proyecto, pero se incluye también aquí por completez: `http://bioroboticsunam.github.io/pyRobotics/`.