Universidad Rey Juan Carlos

VisualHFSM 5.0

ROBOTICA

JdeRobot

Samuel Rey Escudero

samuel.rey.escudero@gmail.com

30 de junio de 2016

Índice

- Introducción
- Objetivos
- Infraestructura
- VisualHFSM 5.0
- Experimentos
- Conclusiones

Introducción

- La **inteligencia** de los robots radica en su **software**.
- Existen diversos métodos para su programación.
- Programación visual.

- Autómatas de estado finito.

VisualHFSM ha ido evolucionando con el paso del tiempo.

Objetivos

El objetivo principal es alcanzar una versión madura y atractiva de VisualHFSM para que sea utilizada por terceros.

1. Mejorar la usabilidad y funcionalidad del editor gráfico.
2. Recuperar la GUI en ejecución.
3. Generar componentes en Python.
4. Fomentar la difusión.

Infraestructura

- JdeRobot 5.3.2
- Gazebo 5.3
- ICE 3.6
- GTK+
- PyQt4

VisualHFSM 5.0

- Mejoras en el editor gráfico.
- GUI en ejecución para C++.
- Generación de código en Python.
- GUI en ejecución para Python.
- Difusión.

Mejoras en el editor gráfico

- Mejor navegación niveles.
- Función `shutDown()` para terminar la ejecución.
- Más flexibilidad para crear el archivo de configuración.
- Puede ejecutarse desde cualquier directorio.

GUI en ejecución para C++

- Permite observar dinámicamente los estados activos en tiempo de ejecución.
- Aspecto similar al editor gráfico.
- Desactivada por defecto, se activa con el argumento *–displaygui=true*.
- La GUI no depende del XML.
- Opción de *autofocus*.

```
def createAutomata(self):
    guiSubautomataList = []

    # Creating subAutomata1
    guiSubautomata1 = GuiSubautomata(1,0, self.automataGui)

    guiSubautomata1.newGuiNode(1, 0, 112, 126, 1, 'Go')
    guiSubautomata1.newGuiNode(2, 0, 372, 136, 0, 'GoBack')
    guiSubautomata1.newGuiNode(3, 0, 238, 308, 0, 'Rotate')

    guiSubautomata1.newGuiTransition((112, 126), (372, 136), (237, 114), 1, 1, 2)
    guiSubautomata1.newGuiTransition((372, 136), (238, 308), (305, 222), 2, 2, 3)
    guiSubautomata1.newGuiTransition((238, 308), (112, 126), (140, 216), 3, 3, 1)
    guiSubautomataList.append(guiSubautomata1)


    return guiSubautomataList
```

Generación de código Python y su GUI en ejecución.

- Da mayor flexibilidad a la herramienta.
- La plantilla sigue un modelo de OOP.
- La GUI en ejecución funciona igual que en los componentes de C++.
- Además permite visualizar más de un nivel a la vez.

```cpp
int Generate::init_py (){
    this->fs.open(this->path.c_str(), std::fstream::out);
    if (this->fs.is_open()){
        this->generateHeaders_py();
        this->generateAutomataClass_py();
        //CREATE GUI SUBAUTOMATAS
        this->generateMain_py();
        this->fs.close();

        this->fs.open(this->cfgpath.c_str(), std::fstream::out);
        if (this->fs.is_open()){
            this->generateCfg();
            this->fs.close();
        }

        std::string permission("chmod +x " + this->path);
        system(permission.c_str());
        return 0;
    }else{
        return -1;
    }
}

void Generate::generateAutomataClass_py(){
    this->fs << "class Automata():" << std::endl;
    this->fs << std::endl;
    this->generateAutomataInit_py();
    this->generateFunctions_py();
    this->generateStartThreads_py();
    this->generateCreateGuiSubautomataList_py();
    this->generateShutDown_py();
    this->generateRunGui_py();
    this->generateSubautomatas_py();
    this->generateConnectToProxys_py();
    this->generateDestroyIc_py();
    this->generateStart_py();
    this->generateJoin_py();
    this->generateReadArgs_py();
}
```

```python
#!/usr/bin/python
#HEADERS
import Ice
from automatagui import AutomataGui, QtGui, GuiSubautomata
....................
class Automata():
    def __init__(self): ...

    def startThreads(self): ...

    def createAutomata(self): ...

    def shutDown(self):

    def runGui(self): ...

    def subautomata1(self): ...

    def connectToProxys(self):

    def destroyIc(self): ...

    def start(self): ...

    def join(self): ...

    def readArgs(self): ...

if __name__ == '__main__':
    signal.signal(signal.SIGINT, signal.SIG_DFL)
    automata = Automata()
    try:
        automata.connectToProxys()
        automata.readArgs()
        automata.start()
        automata.join()

        sys.exit(0)
    except:
        traceback.print_exc()
        automata.destroyIc()
        sys.exit(-1)
```

Difusión

Además de madurar y mejorar la funcionalidad de la herramienta, para que sea utilizada por terceros también nos hemos encargado de:

- Una documentación web.
- Artículo aceptado en el congreso robótico WAF 2016.
- Práctica Choca-Gira en el entorno docente JdeRobot.

Experimentos

Estos experimentos nos han permitido:

- Probar que las mejoras añadidas funcionan correctamente.
- Mostrar la compatibilidad con un nuevo robot: los drones.

3 aplicaciones programadas en Python:

- Choca-Gira.
- Monitorizar un área.
- Aplicación Sigue Colores con un drone.

Choca-Gira

- Aplicación sencilla resuelta con un autómata mononivel.
- Solución a la práctica propuesta en el entorno docente JdeRobot.

Monitorizar un área

- Comportamiento más complejo.
- Autómata multinivel.
- Prueba la compatibilidad de la interfaz del ArDrone en VisualHFSM.

Sigue Colores

- Los componentes generados son compatibles con robots reales.
- Muestra la potencia del autómata frente a sistemas reactivos puros.

Conclusiones y trabajos futuros

1. Se ha mejorado la usabilidad y robustez del editor gráfico.
2. Se ha añadido la GUI en tiempo de ejecución con jerarquía para C++.
3. VisualHFSM es capaz de generar componentes en Python.
4. Los componentes en Python disponen de GUI en tiempo de ejecución.
5. Se ha realizado un trabajo de difusión para fomentar su uso.

Más de 2500 líneas de código.

- Inspección variables en ejecución.
- Botón de parada segura en tiempo de ejecución.