VECTOR TOOLS

Aravind Vadali
Department of Electrical and Computer Engineering
Carnegie Mellon University
avadali@andrew.cmu.edu

Fengzhi Li
School of Computer Science
Carnegie Mellon University
fengzhil@andrew.cmu.edu

May 9, 2019

ABSTRACT

For the final project of 15-418/618, we developed Vector Tools, a set of user tools for Anki Vector using Vector SDK Alpha. This tool has a command line interface, a Finite State Machine program converter, and supports executing the converted .py file. The Finite State Machine program supports the same syntax as cozmo-fsm. Vector Tools FSMs are backwards compatible with Cozmo Tools, and allow the user to take full advantage of Vector's advanced features and concurrency.

***K*eywords** Anki Vector · Finite State Machine · SDK Tools

1 INTRODUCTION

On October 12th, 2018, Anki released its new generation home robot, Vector. Vector has a similar appearance to Cozmo, but has more advanced features and different SDK APIs. Specifically, Vector comes equipped with a full color camera, a laser rangefinder, internet connectivity, voice recognition, and powerful concurrency features built-in. However, many of the changes and additions to Vector result in the API differing from the Cozmo SDK, which is what Cozmo Tools is based off of. In order to allow future Cozmo-tools users to work with Vector in the same way they currently work with Cozmo, we aim to implement the FSM and StateNode system of Cozmo Tools and add some of Vector's advanced features as new nodes.

2 PRIOR WORK

Cozmo-tools is a toolset that allows users to define Finite State Machines (FSMs), and interact with Cozmo in a simpler, less verbose manner than is possible with the bare Cozmo SDK. Cozmo-tools provides a command line interface (`simple_cli`), many types of nodes to perform various actions (`Forward`, `Turn`, etc), transitions to conditionally move from one node to the next (`Success`, `Completion`, `Failure`, etc), and `genfsm`: a tool to translate an FSM defined in simple domain-specific-language into a Python state machine program.

Vector SDK Alpha introduces new features and interfaces to Vector that were not available in Cozmo. Specifically, a major change includes the addition of `AsyncRobot`. While Cozmo SDK's actions all returned some type of status indicating the immediate outcome of the requested action, `AsyncRobot` returns a Python `future` object, which can hold the status of an action before the action is completed, and can have callbacks attached to it. We note that these callbacks provide an ideal method of conditionally transitioning from one action to the next, and setting the entire FSM up before any node necessarily finishes running.

3 METHOD

Our first approach to creating Vector Tools was to simple take Cozmo Tools and substitute the appropriate functions wherever the Cozmo SDK had changed. Our reasoning for this approach was that upon first inspection, the Vector SDK seemed fairly similar to the Cozmo SDK. We reasoned that this would be the easiest way to provide all the functionality Cozmo-tools provided, since we would leave in everything in Cozmo-tools that still worked.

However, after attempting this approach, we came to realize that the way Cozmo-tools implemented concurrency and transitions conflicted with the Vector SDK's method of implementing the same. Unlike the Cozmo SDK, Vector SDK provided asynchronous features and callback functionality out-of-the-box, and attempting to bypass these features and re-implement them as in Cozmo-tools was frustrating to say the least. Furthermore, many of the quality-of-life and visualization features, such as the cam viewer and worldmap viewer, were given automatically in the Vector SDK, and running both types of viewer (ours and the the SDK's) created significant overhead and lag.

Our second approach was to build up the same nodes, transitions, and FSM system Cozmo-tools implemented from the ground up, using Vector's concurrency features. We do this by redefining `StateNodes` and having each one store its `future` object as its "action". Each `StateNode` is also given a list of awaiting transitions. When an action is started (after the program is processed and the FSM is started), all the awaiting transitions are added as callbacks to the action, to be called when the action is completed. Each transition can have multiple sources and multiple destinations and will be fired if its respective condition is satisfied. Listing 1 shows the layout of the major data structures we use.

struct {
  future action;
  List[Transition] awaiting_transitions;
} *StateNode*;
struct {
  List[StateNode] sources;
  List[StateNode] destinations;
  boolean_fn condition;
} *Transition*;

Algorithm 1: Listing of the StateNode and Transition data structures we use in Vector Tools.

4 RESULT

In the latest version of our Vector-tool, we implemented a simple command line interface, where the users can directly control the robot by Vector SDK API functions or FSM nodes. The interface also supports low battery warning, cube connecting, robot connecting, 3D world viewer and camera viewer.

The users can directly use their cozmo FSM program and the cozmo-tools' `genfsm` to generate the python file. Then they can call `runfsm("filename")` in the Vector-tools command line interface to run the program. For FSM program, we support the following nodes and transitions:

Node List

1. **Forward**(*distance, speed*)
2. **Turn**(*angle*)
3. **SetHeadAngle**(*angle*)
4. **SetLeftHeight**(*height*)
5. **MoveLift**(*speed*)
6. **GoToPose**(*pose, relative*)
7. **GoToPosition**(*x, y, angle, relative*)
8. **DriveOffCharger**()
9. **DriveOnCharger**()
10. **Say**(*text*)
11. **TakePicture**()
12. **DisplayImageOnMonitor**()
13. **DisplayImageOnScreen**()
14. **MirrorMode**(*enable*)

Transition List

1. **CompletionTrans**() `=C=>`
2. **SuccessTrans**() `=S=>`
3. **FailureTrans**() `=F=>`
4. **NullTrans**() `=N=>`
5. **TimerTrans**() `=T(x)=>`
6. **DataTrans**() `=D(x)=> / =D=>`

The source code of Vector-tools can be downloaded here: `https://github.com/L-F-Z/vector-tools`

5 EXAMPLE

For public demonstration, we wrote the following FSM program to show the capability of Vector-tools.

```
from NewFSM.NewFSM import *
class SimpleMotion(StateMachineProgram):
    $setup{
        Forward(50) =C=> Turn(30) =C=> {driver, speaker}
        driver  : Forward(-50) =T(5)=> Say("All Done") =C=> SetHeadAngle(degrees
            (45)) =C=> photo
        speaker : Say("Save Anki!")
        photo   : TakePicture() =D=> DisplayImageOnScreen(5) =T(5)=> mirror
        mirror  : MirrorMode() =T(5)=>MirrorMode(enable=False)
    }
```

We recorded a video for this demonstration: `https://youtu.be/ASuBbNdW16s`

6 ACKNOWLEDGEMENTS

We would like to acknowledge Professor Dave Touretzky and TA Mandy Huang for their guidance and support throughout this semester.