Tracie Report

Team 22 [*]

November 27, 2014

[*]Members: Mitchell Kember, Do Gyun Kim, Charles Bai, Xiao-Yang (Michael) Min, Min Suk Kim, Renato Zveibil, Leong Si

Table of Contents

1 Executive Summary

"Tracie" utilizes the Scribbler Bot and the Fluke2 chip to trace out the drawable input transferred from the user interface, via a web app, through linear movement and precise rotations. Programmed in JavaScript, the client, which runs in the browser, is the ultimate form of communication between the user and the robot. The client provides the user the ability to start and stop the program immediately, and allows the user to modify the data values of the Controller, which alters the distance and time conversion and rotation to time conversion for optimization purposes. The user interface, integrated within the client, allows the user to input a set of points and provides backwards and clearing functionality. When the program runs, the client communicates with the Server and Controller, both simultaneously running in the terminal, to start up the Tracie script and retrieves information influencing the robot's performance, such as the speed. The Tracie program is coded in Python using the Myro library [1], a Python-written framework used for programming robots. The Tracie program supplies the artificial intelligence and program logic to manipulate the Scribbler Bot's movement.

This report contains a Work Breakdown Structure (Figure 1), a PERT Network Diagram (Figure 2), and a Gantt Chart (Figure 3). They provides a strong foundation in optimizing the time spent on each activity through an organized schedule. Additionally, this report includes a requirements document, which identifies the personnel in the group and their respective roles and contributions, and lists the functional and non-functional requirements in order to progress and complete the project. Furthermore, a computation decision making (CDM) section will also be included. CDM is essentially the description of how the problems in the project are being analyzed, as well as the number and description of the criteria and alternatives found. The project retrospective section will include what worked well and what did not in regards to the supplied resources, (such as the Scribbler Bot or Fluke2 chip), team organization, and the development process.

2 Requirements

2.1 Roles

Mitchell Kember: Project leader, coordinator, and organizer. Mitchell set up Trello boards [2] to keep track of progress and objectives for the project. Additionally, he set up the client and server [3][4] on which Tracie is run on and contributed to the majority of the code.

Do Gyun (Justin) Kim: User interface designer. Justin created the user interface and its functions with JavaScript.

Charles Bai: Algorithm developer and reporter. Charles created algorithms for rotation to time conversion and distance calculations. He also wrote the Executive Summary, designed the Planning Diagrams, and worked on the Requirments part of the report.

Xiao-Yang (Michael) Min: Manager. Michael verified when objectives should be completed in order for the project to be completed on time.

Leong Si: Reporter and report designer. Leong was responsible for the publication of the report. He also worked on the Requirements, Computation Decision Making, Project Retrospective part of the report.

2.2 Functional Requirements

Server: A local host server which allows communication between the user interface on a computer and the Scribbler Bot.

Controller: A program written in Python to control the robot via the server. It should be capable of controlling the robot's movement with different input values.

Math Utility Calculations are needed to be done for the robot to move in the correct speed, angle, and distance. Those calculations will be used to implement the movement algorithm once the calculations satisfy the expected result.

Movement Algorithm Using Math Utility, the algorithm should translate the desired shapes or drawings into a sequence of robot movements.

User Interface The user interface should allow the user to select or create drawings within itself.

Graphical Design The layout of the cilent, which contains the Controller and user interface, should be clean and readable. It should display the current status of the robot, allow the user to change values of the Controller, and have buttons for functions like reset or stop.

2.3 Non-functional Requirements

Constraints

Time

The project must be finished by November 20th.

Money

The team will not receive any payment during or upon finishing the project. We also did not want to spend extra money on this project, so resources are limited.

DC Motor

The DC motor of the Scribbler Bot made it very difficult to do fluent movement, especially turning. Since we are restricted to use only the Scribbler Bot, many more calculations and considerations are required.

Bluetooth

The range of the bluetooth device in the Fluke2 chip is limited. It also has a noticeable delay when performing real-time tasks. Different methods are going to be used to bypass these problems, for example, a web client to extend the range.

Criteria

· **Accuracy of the shapes**

· **Time required to trace**

· **Feasability**

· **Flexibility**

Details of the criteria will be on the next section.

3 Computation Decision Making

After reading the proposals, we came to a conclusion that we should utilize the drawing feature of the scribbler bot. We decided to create a program which allows a user to input a drawing from a computer. The drawing would then be drawn by the scribbler bot on a piece of paper.

There were many criteria we needed to consider before choosing our final drawing method:

1. **Accuracy**

The drawing created by the robot must follow very closely to the drawing on the screen.

2. **Time**

   We wanted the robot to take as less time as possible to trace out the drawing.

3. **Feasability**

   Some of our members had little to no experience in programming, especially in Python. We wanted to share our workload equally, so the project should not be too difficulty for anyone.

4. **Flexibility**

   Our robot would create different drawings based on the user's choice.

With the criteria listed above, we discussed about the possible methods and came up with 3 alternatives:

1) Fixed Shapes

Allow the users to choose shapes from the user interface. After they have selected their shape, the robot will then draw it.

2) Pixel Based

The users will create a pixel based drawing on the user interface, then the robot will recreate the drawing on a piece of paper.

3) Point to Point

The users will insert “points” on the user interface. Each point is connected directly to the point placed before it. The robot will then follow the points in order, thus create a drawing.

Table 1: CDM of Tracie

Criteria	w_i (%)	Fixed Shapes			Pixel Based			Point to Point		
		C_{i1}	r_{i1}	S_{i1}	C_{i2}	r_{i2}	S_{i2}	C_{i3}	r_{i3}	S_{i3}
Accuracy	30	10.0	1.0	0.30	9.0	0.9	0.27	8.0	0.8	0.24
Time	10	10.0	1.0	0.10	3.0	0.3	0.03	7.0	0.7	0.07
Feasability	30	10.0	1.0	0.30	5.0	0.5	0.15	9.0	0.9	0.27
Flexibility	30	1.0	0.1	0.03	10.0	1.0	0.30	9.0	0.9	0.27
				0.73			0.75			0.85

The result from Table 1 showed that the Point to Point method satisfied majority of the criteria, therefore became the method of tracing for our project.

4 Project Retrospective

4.1 Resources

During the production of Tracie, we faced different types of problems, especially the ones which involved hardware. The robot provided, the Scribbler Bot, were not equipped with well-functioning hardware. We could have utilized the sensoring devices of the robot for the project, but decided against it since they were not reliable. Instead, we worked with the most consistent parts of the robot, the motor and the drawing “function”.

The Fluke2 chip was also a supplied resource. Similar to the robot, the sensoring devices could not provide the accuracy we needed. Also, as discussed in Section 2.3, the bluetooth connection was a major problem throughout the project.

The Myro library was used in the codes to access parts of the Scribbler Bot. It contained all the functions needed to control or received information from the robot. It made coding for the project very simple. However, the only problem which we struggled with was to turn the robot to a specific angle. The functions in the library allowed us to turn the robot or rotate one of the wheels for specific amount of time, but no angle.

4.2 Project Organization

The final product, Tracie, was consistent and accurate. We were very satisfied with the result of the project. The project allowed us to learn and practise the development of software, as well as seeing it in action as a physical unit. This project served an excellent educational purpose while being interesting to those who contributed.

The development process of the project went surprisingly well. All the components needed to progress were finished before planned. Thanks to Mitchell, who set up GitHub and Trello boards, we were able to clearly identify the critical paths and important dates.

Although the project itself went very smoothly, the organization within our team did not. Since we are all first-year university students, we had trouble managing our schedules. The result was an unbalanced workload distribution within our team. While some of our members attended all meetings and participated in all aspects of the project, others did little to nothing.

4.3 Solutions and Alternatives

```
content = raw_input("Seek help from Mitchell.")
print "%s" % content
```

5 Diagrams

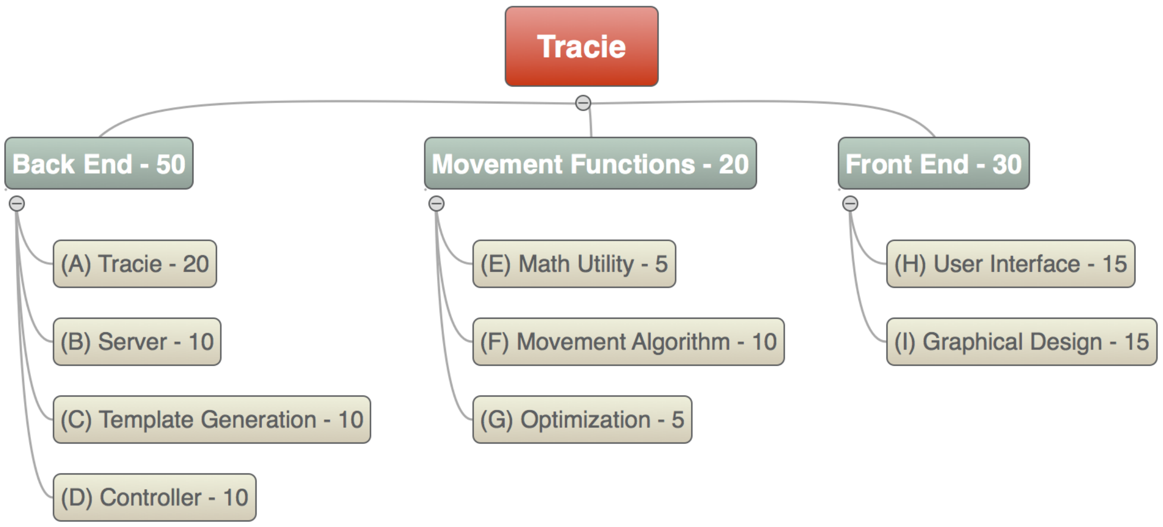

Figure 1: Work Breakdown Structure of Tracie

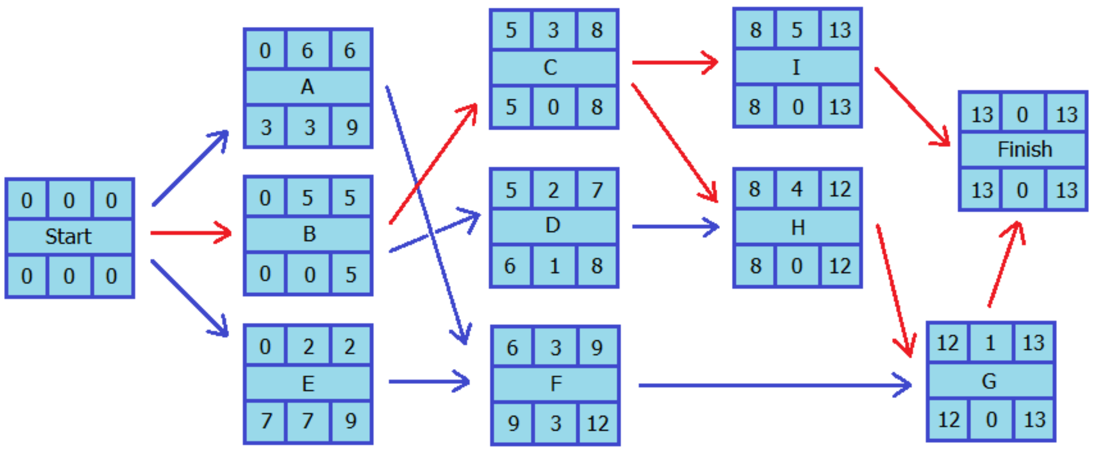

Figure 2: PERT diagram. Critical paths: {B,C,H,G}, {B,C,I}.

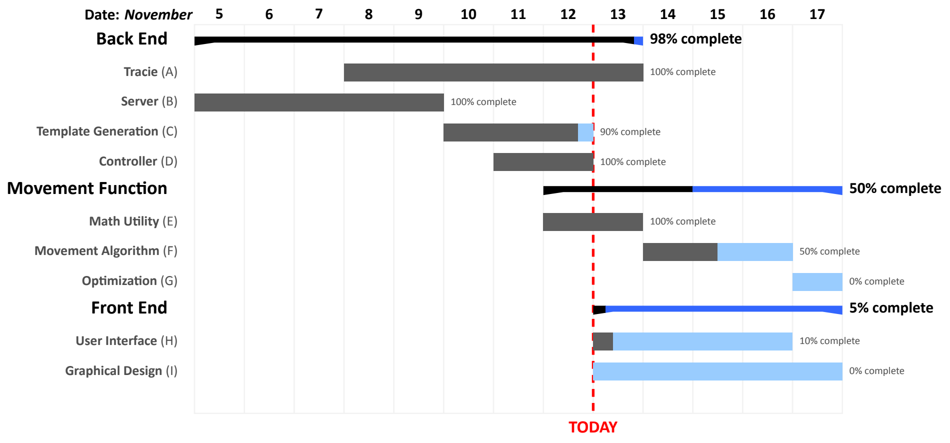

Figure 3: Tracie's Gantt Chart on Nov. 13, 2014

6 Reference

[1] “Myro Reference Manual,” wiki.roboteducation.org/Myro_Reference_Manual. Accessed November 18, 2014.

[2] M. Kember, “Tracie,” trello.com/b/iJbcFOOv/tracie#. Accessed November 20, 2014.

[3] D. Bilenko, “Gevent: A coroutine-based network library for Python,” www.gevent.org. Accessed November 14, 2014.

[4] M. Hellkamp, “Primer to Asynchronous Applications,” www.bottlepy.org/docs/dev/async.html. Accessed November 14, 2014.