DIT524 V15 Project: Systems development

Alice SmartCar

Post Mortem Report

Group 5

Kai Salmon	960421-T012	kaisalmon@hotmail.com
Martina Freiholtz	900501-3405	m@rtina.be
Tobias Lindell	920104-5318	tobias.lindell@hotmail.com
Rachele Mello	880826-0809	rachelemello@gmail.com
Linhang Nie	930317-9593	kevin.nie@hotmail.sg

May 29, 2015

Contents

1 Introduction

Alice is a small robotic car built using an Arduino module together with other components. The car itself was the product of previous work done by our supervisor Dimitris Platis and his group. Based on the starting point of the physical car and its Arduino libraries, our group developed an Android application for portable devices in order to control it. We implemented two control modes: Alice follows a path that was drawn directly on the screen of the portable device, converted into executable instructions and received by bluetooth, stopping automatically in case of obstacle detection; or Alice can be manually drove around controlled by a virtual joystick. Obstacle detection is done with the help of an ultrasound sensor mounted on the vehicle. More advanced features of the app include the possibility of uploading an image as background when drawing the path (as a blueprint), scaling and different degrees of simplification of the drawn curve.
The potential applications of such product might be in large industries, where automated vehicles might be used in warehouses, or in exploring potentially dangerous areas.

The project was carried out mainly in the second part of the semester. The initial phase required us to get familiarised with Arduino itself and its possibilities. Not having experience in the area, we did not know what the potentials and limitations of it were, therefore it took us a while to define our product vision. We had a few initial meeting with our supervisor in which we "played around" with Arduino, coded some simple sketches and leared some basics regarding the sensors.

In the second section of this report, the main processes and practises that we used, within the big picture of a SCRUM approach, are discussed.
In the third part, we try to critically look at our work and analyse its strength and weaknesses and reflect on the future.
Finally, conclusions are drawn. The appendix section contains details regarding different aspects of the development.

2 Processes and Practices

More than any other project that we have worked on, this one has created a development experience that is most similar to how development will be when we enter the industry. Before any actual product value could be created, we needed to acquire the skills needed to expand on the car that was created by our supervisor. For example, we needed to learn the language and techniques for programming the Arduino chip, as well as learn the capabilities of the methods in the API's library.

This slow start to product value lead the team members to specialize in different ways to maximise the value of the time spent on development. Rachele, Kevin and Martina became specialists of Arduino, while Tobias and Kai focused on the Android application and the custom views that were required. This also provided us with a great way to organize our scrum sprints. For every sprint meeting, the two teams coordinated their efforts and eventually combined their efforts on the same car.

Our team used Pivotal Tracker to keep track of many aspects of our development efforts. Pivotal Tracker is a tool that logs any and all tasks that the team has decided are necessary to complete the project. These tasks are well defined and can be freely moved from the “icebox” into “current”. The tasks in the current category will be assigned to specific team members as well as contain a precise definition. It also helped define exactly what was needed for a task to be considered done.

Pivotal was very helpful in keeping track of which set of group members worked on which task, so there was no unnecessary work done. For example, if a team member was finished with the task that was assigned to him or her, he or she could use pivotal tracker to find out exactly who was still working on their task. We also used pivotal tracker to measure the amount of technical debt we were in, and to keep track of if we were keeping up with the intended velocity. In the later stages of development, we added smaller features to the application. When doing so, we could easily add and complete a task on pivotal tracker to keep track of exactly when we completed what part of the project.

2.1 Technique 1: Continuous Integration and Incremental Development

Continuous integration and incremental development are two agile techniques that worked hand in hand for our team. Continuous integration in its essence means that you merge changes to a master multiple times a day. Our development did not work this way exactly because of how changes were not made that frequently. However, when the development team worked together for extended periods, each change we made could be tested and integrated into the system. Because we had the car nearby, we could easily evaluate if the changes made worked like they should. Constantly checking how the changes worked with the rest of the system was a good way to discover bugs.

Using incremental development to structure the development effort was a very organic way to make continuous integration work properly. In order to use the changes on the car, we needed to start on increments that would produce a usable car. For example, on the Arduino side, as soon as the Bluetooth connection was functional, we prioritized creating a protocol for string parsing so that the car would understand the string list sent from the Android application. Even though obstacle detection and other features like that were not developed at all, we could test them as they were created on our car which had only basic functions. This meant that we fully developed each feature as it was added to our current tasks, and didn't move on to the next until it was finished.

2.2 Technique 2: Pair Programming

Because of the nature of some tasks, it was hard to work on them from two computers. Thus, it was easier to use pair programming to make progress. When creating features like the settings menu, it was easier to do the main coding on one computer, while another group member did research on specific aspects of the process like which type of Android preference to use.

Pair programming was particularly useful when bugfixing. One person could send test commands to the car while the other made sure the car didn't crash in a way which would cause damage it. Later both team members could work together in researching the errors and make the necessary changes on one computer.

The setback of using this technique was that there were some tasks that were on ice because they depended on the completion of other features. It means that pair programming probably wasn't the most efficient way to make progress if working on multiple tasks was an option. For example, the Android end of collision detection was a high value task and required many hours to complete. However most of the features could not be completed without a working version of collision detection on the Arduino board. Even though the command processing and the collision detections functioned separately, they were not working at the same time until much later in the development process.

2.3 Technique 3: Story Driven Modelling and Refactoring

Instead of static class diagrams, we made most our design choices considering use cases. The main user story we referenced is one where our product is being used in a storage facility. Many features like the scaling of settings alternatives were decided with this user story in mind. This mindset was very beneficial when working on the later stages of the application. We were flexible in adding classes and changing the structure of the system to fit the user stories. For example, when adding the manual drive mode, we created a new class that would handle the changing layout of the main activity dynamically. For this feature to work smoothly, we had to refactor some code.

Although this refactoring made the code more efficient, we would not have needed to put in the time to refactor at all if we had more structured class diagrams ahead of time. We still feel this was not possible for us, however, because of how many unknowns there were before starting development.

3 Project Discussion

3.1 Vision vs Final product

Reading our initial Product Vision (Appendix A), one can notice some substantial differences with the final result: Alice is manually controlled, but not through tilting of the phone, and it does not drive autonomously recreating a graphical representation of its surroundings.

The reason for these differences mainly lays in the fact that we were not familiar with the Arduino world and its possibilities. It was hard for us to estimate beforehand what could have been considered feasible or not for this kind of project.

Once we realised our potentials and limitations, it was easier to come up with a general vision of what we wanted to deliver. This vision then adapted and evolved, based on our reflections regarding the possible stakeholders and their needs and expectations. Some features added on our initial idea, while other got lost as we analysed them as a lower priority or not actually an added value to the product.

As an example, we initially put quite some thought into how to position, control and integrate different sensors for obstacle detection (no sensors were present initially on the car). We had considered combining a steady ultrasound sensor with an infrared one mounted on a rotating servo. We believed that both were needed because we could not rely on the infrared one for accurate distance measuring, nor on the ultrasound for detecting obstacles on the front as its vision is rather wide. Some testing revealed that the ultrasound sensor was instead accurate enough and we realised that we did not need great precision in the prospect of a factory warehouse, where obstacles would be rather large and probably steady.

3.2 Accomplishments

When the group members became specialized in their respective tasks, we did a very good job in making the different sides of the system work together. During the weekly sprint meetings, we agreed on translation protocols for the Arduino and Android sides to use to communicate with each other. These protocols held and made for very fast results once the bluetooth communication was sending information successfully.

Another thing that we did well was to catch up after progress was slow in the earlier stages of development. There were a few sprint meetings where we were not crossing many things off the list, and tasks were not being completed as quickly as we had hoped. However, we successfully picked up the pace to compensate for this, and was not pressed for time at any submission date.

3.3 Difficulties

Even though our group did not execute some aspects of agile development perfectly, we believe that a realistic implementation of an agile development process does not thrive in every way possible. For example, although we did structure our development schedule after our sprint meetings, we did not use planning poker or anything similar to place different levels of importance on the tasks. This meant that our planning on the long term was a little weak. With the help of planning poker, we could have started with the tasks that had the highest product value to offer, while we instead just took on the tasks that seemed iteratively logical. For example, the bluetooth connection was developed a little later than we would have liked, since it presented with a large difficulty. Instead we started with designing specific views for the Android application, as well as developing code for the Arduino that uses the sensors.

To give the team a better continuous understanding of our progress, it might also have been a good idea to add scheduled development efforts between the sprint meetings. Instead the subgroups worked on their tasks sporadically, and the technical debt of our development effort fluctuated. By the last couple of sprints, we were not pressed for time, however, so we were able to catch up and negate the technical debt. Pivotal Tracker logs will be able to show exactly when each task was assigned and completed.

3.4 Group dynamics

Throughout the creation of this project, the members of the group fell into certain roles of varying amounts of responsibility. Tobias and Kai fell into leadership positions and took some design decisions unilaterally, while the other members got assigned tasks during meetings and completed them according to specifications. For example, while implementing the manual drive mode, Tobias decided on how to handle the new requirements on the bluetooth connection, as well has how to handle the new view that was required. Kai designed the visuals and functions of that manual drive view so that it would do its job, and also fit into Tobias's structural design.

The group had some trouble with handling a group member not pulling their weight. Even though progress on development was not directly slowed down because of this member, it was troubling to the group to deal with one member who had no computer, and could not contribute to any part of the project. Up until the point of writing the final report, the group was under the impression that they were not registered to the course. In the end, we could not with good conscience put their name on the finished product.

3.5 Reflections for the future

When participating in future projects, we would probably make an effort in creating tasks for individuals, rather than for groups. This would create a more tangible way for each member to contribute equally. It is still a good idea to have some of those tasks work organically together so people can work together. Pivotal tracker is an agile tool that we would easily use in future projects. It is a very clear way to keep track of tasks and to put more emphasis on the ones which are more important. However, when using pivotal tracker in the future, we will put much more effort into properly putting information into the tasks at the point of marking them as complete. The pivotal tracker backlog is our main source of information for tracking our progress, and people's contribution, but in many cases, the wrong names are attached to the wrong tasks because they were completed by different group members.

4 Individual contributions

The contributions of each team member can be seen from pivotal tracker. To start off, team members were given the tasks to research into different aspects of our development, like the functionality of bluetooth, arduino, and android. The people that were assigned to research something had responsibility for that side of the system for the remainder of the project with the exception of one member.

Tobias has been responsible for most things regarding the bluetooth connection while helping Kai with some algorithms in the Android application. Kai has created the custom views, most other functionality in the Android app, as well as the first versions of the Arduino sketch which carried out a hard coded sequence of commands on the car. Kevin and Rachele have been responsible for designing and implementing the structure of Arduino sketches which interact with the app to control the car. Martina contributed to the Arduino sketches as well as being in charge of designing the GUI.

5 Conclusions

In addition to testing us in the ways of developing systems, this project was also testing the accessibility of the car's library, and if it was easily used by people who has no prior knowledge of it. We believe that our supervisor has created a very well structured system. With just two short Arduino lessons and some research, we were able to create a system with so many real life uses on the shoulders of his work. Even though it felt like a daunting task in the beginning, it has been a very informative and confidence boosting experience which has turned us into more competent programmers.

Appendix A - Product vision

Initial vision

Our product is a robotic car, remotely controlled with a phone application. The target customers for our product at this stage are individuals and groups interested in robotic technology, as a hobby product or for scientific purposes. We also envision the product being used for more specific purposes, such as the exploration and mapping of unsafe areas. The product could be of assistance to military organisations and law enforcement in high-risk situations, such as bomb extractions.

The following two functionalities will help to meet the users' needs:

1. The user will be able to control the car manually through a phone application, at the first stage available on the Android operating system. There will be an option to control the car by simply tilting the phone in the direction the user wishes the car to go, adding simplicity and immersion. This is made possible by the phone's built in sensors.

2. The car will be able to drive autonomously while sending information about its path and a graphical representation of its surroundings to the phone application.

The latter feature in particular sets this product apart from many similar products at this level.

Updated vision

The project consists of two components:

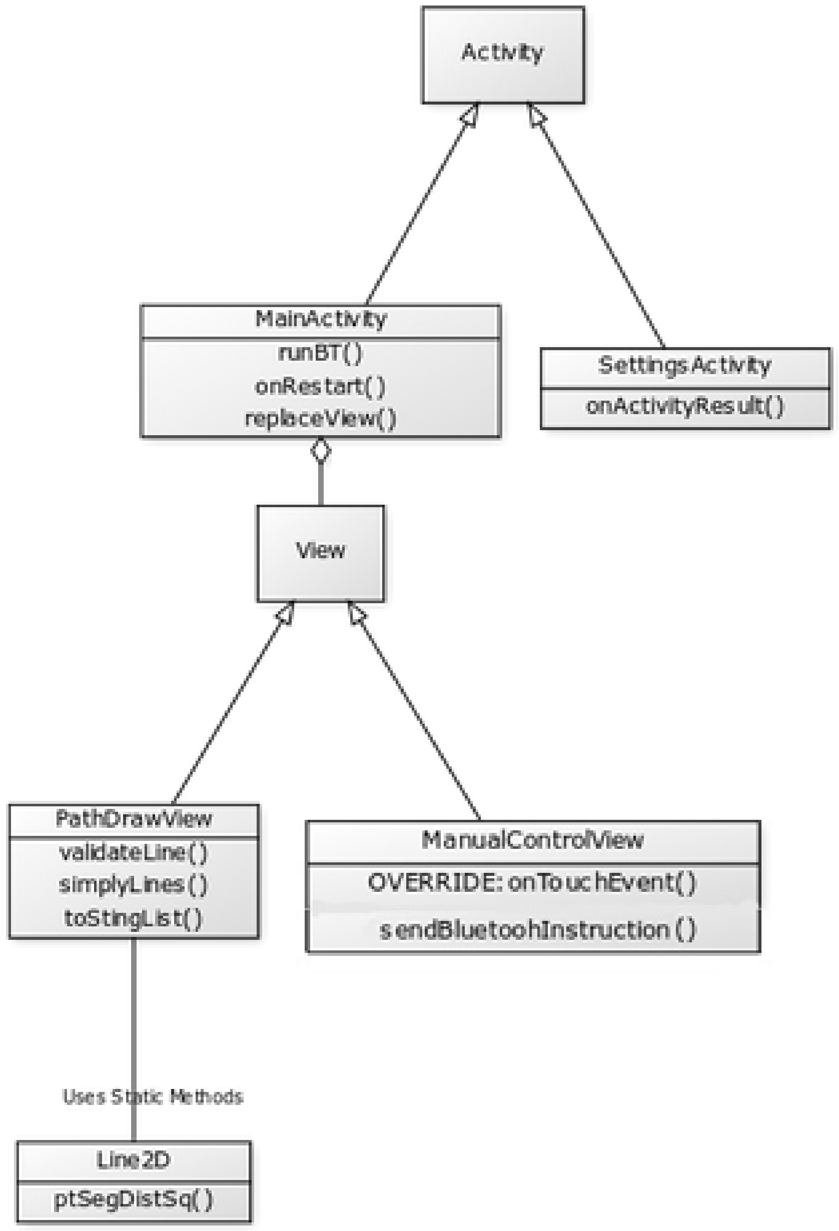
1. An autonomous car, programmed on the Arduino platform.

2. An application for a portable "smart" device (a smartphone or a tablet computer), currently running on the android platform.

The user draws a path directly on the screen of the portable device for the car to follow. The car and the portable device communicate via Bluetooth. The car is equipped with an obstacle-detecting sensor (currently ultrasound). Should an obstacle be detected, the car stops and the user is presented with two possible choices: either control the car manually, or instruct the car to carry out previously sent instructions if the obstacle is deemed to not be a threat.

The user can upload an image to the application and draw the path directly on top of the chosen image. This simplifies the process of drawing a more precise path over a larger area.
The software could be incorporated into alternate hardware for specialized and repetitive tasks, such as moving stock through a factory or warehouse. There is also a potential to use the car for similar tasks in areas and environments which are unsafe for humans.

Appendix B - Developer Documentation: Android



Main Activity

The Main Activity contains all the views and functionality that the user will interact with. Along with the standard button views, there is the pathDrawView, and the manualControlView, which will be described below.

In the MainActivity, there are a couple important methods that will need to be understood before changes can be made to the system.

runBT()

The runBT method gets called when the user presses the connect button. It creates new instances of everything that is needed to make a connection between the android device and the arduino's bluetooth module.

There is a single static field, the OutPutStream, which is mentioned from both the pathDrawView and the manualControlView.

onRestart()

On restart is important to the MainActivity because it gets called every time the user enters the activity from the settings menu. Any changes the user made while in the settings will be revalidated automatically to show the changes.

replaceView()

replaceView is called whenever the swap button is pressed. It efficiently handles the visibility of the pathDrawView and manualControlView so that the structure of the activity remains the same when they are switched out.

Path Draw View

validateLine()

The Validate Line method is called whenever the users lifts his finger from the screen. indicating they have finished drawing their path. It runs a number of methods to improves the line quality, such as removing duplicate points. It also calls simplifyLines() and toStringList().

simplifyLines()

The Simplify Lines method is called by ValidateLine(). It uses and implementation of the Douglas-Peucker algorithm to remove points from the line which do not contribute to the overall shape. Depending on the settings chosen by the user more or less points are removed by the algorithm

toStringList()

This method is called by validate line. It reads the list of points given by the user when they drew the line, and converts into an instruction that the arduino car can carry out. It does this by looking at every set of three adjacent points (That is, every pair of adjacent line segments) and calculating: The length of the first line segment, in regards to the scale of the draw view defined by the user The angle between the two line segments, and if that angle is clockwise or anti-clockwise These instructions are compiled into an array of Strings, such as:

```
goForward 50*
rotateClockwise 90*
goForward 30*
rotateCounterClockwise 45*
goForward 20*
```

(These would be the instruction for a line with three line segments, and four points)

Note: The angles are in degrees, and the distances are in centimeters.

ManualControlView

sendBluetoothInstruction

This method sends single strings as instructions through the static OutputStream field that was created in the MainActivity. The commands are designed to make the car execute that instruction until a new one is given, or until the joystick returns to the center and sends a "stop" instruction.

onTouchevent()

This method overrides the parent method. If the touch event is a Move or a Down event, then the method checks the region of the screen that the user is pressing, and if relevant, sends a manual control bluetooth command to the car. If the touch event is an up event, then the stop command is sent to the car.

Settings Activity

The settings activity is structured in a way to allow for future modification. The settings activity implements an instance of SettingsFragment. A new settings fragment can be created and used by the current settings activity without having to make any changes to the system itself.

onActivityResult()

The onActivityResult method handles the result of a selection made from the phone's external storage. This happens after the user has pressed the change background button. When the user chooses a valid image, the onActivityResult rotates and resizes the image when applicable, and sets it as the background.

Appendix C - Developer documentation: Arduino

About the system

Basically there are two main parts of the application: the Android app and the Arduino program. The Android app mainly handles the path generating, manual control function, and the corresponding algorithms. Arduino part deals with the issue of reacting to the app instructions. The two parts are interfaced with each other by bluetooth 4.2.

About the build process

Generally we follow an iterative process while building the Arduino sketches. After getting basic knowledge about how Arduino works and how to program the board, we started from doing some experiments. Like we designed serveral sensor layouts and validated their effectiveness as well as explored how the Arduino board communicate with ouside via serial.

Each of our functional unit has experienced three main stages, designing, testing and integration. First, we modeled the behavior of the unit and the interactions with actors, then implemented a highly abstract sketch to test with simple input and output. After, we developed the function on detail, and test it as a complete component. Finally the components were integrated to the main sketch.

Design details

API

We use an external API which handles the basic function of the car and the sensor, such as move a certain distance toward a direction, and return distance value when the sensor detect an object. You can find information about the Smartcar and its Sensors from github.

Models

All the design manuscript of models of the sketch and its key component can be found in the Arduino manuscript folder on git repository. They are:

state diagram
model how the car system's state switches to react to internal and external events

sequence diagram
shows the message interactions between the app and the car system components

main route flow chart
illustrates the general structure of the sketch

automode flow chart
: shows the work flow of the automode

key functions flow chart
: include goForwardSafe(int), and isFrontClear() in detail…

The implemention details could be found on git repository.

Optimization

Message transmission and interprete overhead
: In the case of instruction becomes more complex, for instance robotic arm and other component might need to be controled while moving, the message processing unit may need to be optimized. Since longer messages require more CPU and memory to transmit and receive and complex messages require a parser and a string transformer for them to exhibit intended meanings, it will cause larger transmission and interprete overhead. To optimize runtime performance, message length could be minimized and message meaning need to be maximized and the way of processing the message could also be refined and reduced to minimize overhead.

Inter-module coupling
: The car system, especially the message processing unit is relatively tightly coupled. The executing unit relies much on the parser. If the message is to be refined or the parser need to be changed, the executing unit has to be changed. We tried to lower the coupling level by only allowing the components share and pass global data, since the atuality is acceptable, we put our effort into developing other features. But this structure could be optimized to improve the readability and maintainability of the code.…

Appendix D - User manual

Getting started

Installing the application

Download the APK directly to the Android device, or transfer it from a computer over USB. When the APK is on the device, use a file manager application to find the file and install the application on your phone.

Setting up

Make sure the Android device is paired with the vehicle. Do this by accessing the Bluetooth settings on your phone. Make sure that Bluetooth is enabled and the vehicle is turned on. Find the vehicle in the list of available Bluetooth devices, and pair it with the Android device by selecting it.

Open PathCar. The application always starts in path mode, see below. To connect the application to the vehicle, tap ‘CONNECT’.

Path mode

The application always starts in path mode. A path can be drawn directly on the screen with a finger or a stylus. When a Bluetooth connection is established and a satisfactory path has been drawn, tap ‘SEND’ to send instructions to the vehicle.

The path is drawn from the perspective of the vehicle, and thus the first move executed in path mode will always be forward. If the physical space does not allow for this, swapping to manual mode might be necessary, see “Manual mode”.

Delete exisiting path

To delete an existing path, tap the drawing area once. Unless the existing path covers a significant area of the screen, it is not necessary to delete the old path before drawing a new one. When a new path is drawn, the old path will be discarded.

Path correction

A drawn path can be corrected from a certain point, unless ‘Sensitivity of line modification’ is turned off (see “Settings”). Place a finger or stylus close to the existing path (how close can be determined in the settings, see “Sensitivity of line modification”), and draw the desired path from that point. The unused part of the path will be discarded in favor of the new one.

Auto-loop

If the starting point and the ending point of a path is close together, the application can be made to connect these points and create a loop, causing the vehicle to return to its starting point. To disable this function, see “Sensitivity of auto-loop” under “Settings”.

Obstacle detection

If an obstacle is detected while the vehicle is following a path, the vehicle will stop and the part of the path where the obstacle was detected will flash red on the screen. When the obstacle has been removed, draw a new path. Alternatively, swap to manual mode to navigate the vehicle around the obstacle (see ”Manual mode”).

Manual mode

Swap to manual mode by tapping ‘MANUAL MODE’. Note that the vehicle has to be connected for this to work. The mode can be swapped back any time by tapping ‘PATH MODE’.

Navigate the vehicle by moving the digital joystick. Controls:

UP = move forward

DOWN = move backward

LEFT = rotate left

RIGHT = rotate right

Settings

To enter the settings menu, tap the menu icon (three vertical dots in the top right corner of the application) and tap ‘Settings’.

To exit the settings menu, use the back button of the device.

Simplification sensitivity

This setting determines how much the application simplifies the drawn path, if at all. The simplification sensitivity is initially set to ‘Low’, and can be changed to ‘High’ or turned off completely.

Sensitivity of auto-loop

This setting determines how close together the starting point and the ending point of a path need to be in order for the path to be set to a loop. The sensitivity can be set to ‘Low’, ‘High’, or turned off completely (in which case the path will never be set to a loop).

Sensitivity of line modification

This setting determines how close to the existing path a finger or a stylus has to be placed in order to modify the path. The sensitivity can be set to ‘Low’, ‘High’, or turned off completely. In the last case, the old path will always be discarded once a new path is being drawn.

Scale of driving area

This setting determines how wide the physical driving area is in correlation to the screen of the device. The width is set in centimeters, and the application calculates height based on that measurement and the screen area.

Background

To navigate the car with more precision, a background (such as a map or a blueprint) can be set to represent the physical driving area.

Select a background image

Tap the ’CHOOSE’ button to open an image stored on the device. The image will be set as a background in path mode.

Reset background

Tap the ’RESET’ button to clear the background.