ECSE 211 - Design Principles and Methods

SYSTEM DOCUMENT

GROUP 11
John, Durham, Alex, Ian, Ethan

Version 2.02
29th March 2017

Abstract

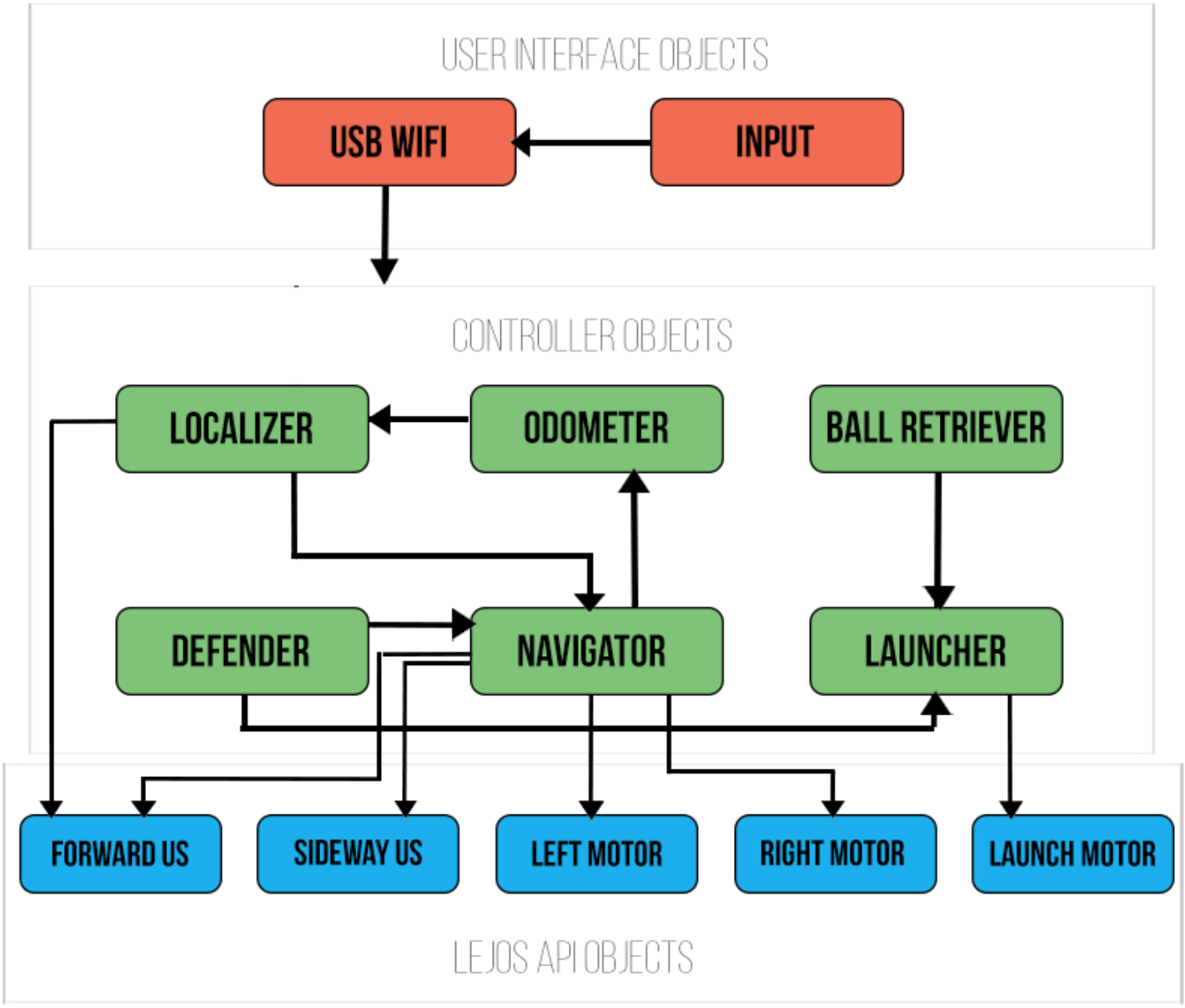
This document is intended to identify and define the solution environment (i.e. the tools and components available for use in the development of a design satisfying the given requirements). This document identifies the capabilities of all parts available for the construction of the final product both software and hardware. This document will outline the components necessary to the development of a solution to our design problem.

Contents

1 EDIT HISTORY

- February 19th 2017:
  - **Ian Smith:** Initial set up of document sections (including table of contents)
- February 20th 2017:
  - **Alex Lam:** First draft of section 10
  - **John Wu:** First draft of section 2 (with Ethan), 4, 6, 7 (including 7.1, 7.2 and 7.3) and 8 (including 8.1, 8.2, 8.3, 8.4, 8.5 and 8.6)
  - **Ethan Lague:** First draft of section 2 (with John) and 3 (including 3.1, 3.2 and 3.3)
  - **Ian Smith:** First draft of section 5 (including 5.1 and 5.2)
  - **Durham Abric:** Reference in section 9
- February 22nd 2017:
  - **Alex Lam:** General edits to formatting and changes to sections 5.1, 5.2 and 10
- March 29th 2017:
  - **Ethan Lague:** Added to section 3.1 and 3.2, changed 8.1, 8.2, 8.5, 8.6.
- April 7th 2017:
  - **Alex Lam:** Added to sections 3.1, 3.2 and 3.3
- April 8th 2017:
  - **Alex Lam:** Added to sections 6, 7.1, 7.2, 7.3, 8.1.1, 8.1.2, 8.1.3, 8.2.1, 8.2.2, 8.2.3, 8.2.4, 8.2.5 and 8.2.6.

2 SYSTEM MODEL



3 HARDWARE AVAILABLE AND CAPABILITIES

3 Lego EV3 "Mindstorm" kits:
These kits include various structural pieces, connection pieces, bars, bar caps, wheels, gears, tires, tracks, 12 large motors, 3 medium motors, 3 ultrasonic sensors, 3 rechargeable battery packs, 4 AA batteries, 3 color sensors, 3 gyroscopes, 6 touch sensors, 3 EV3 bricks, 3 USB cables, numerous RJ12 cables, 3 chargers, and various other specialized Lego pieces. In addition elastic bands of different sizes and elasticity are also available for use.

3.1 Mechanical Limitations

There will be little to no limitations due to availability of building material since the 3 Mind storm kits provided contain more than enough pieces for this project. The mechanical limitations will instead come from the quality, capabilities and versatility of the available pieces. Indeed, due to the nature of Lego, there is only a finite number of possible designs as the Lego pieces can only be connected to each other in a limited amount of different ways. This will be the main mechanical limiting factor as this will restrict the placement of the different components (i.e. motors, sensors, etc.).

The sturdiness of the design is also limited by the spacing between the different Lego pieces as well as the flexibility of the pieces themselves. The plastic Lego pieces are indeed prone to bend under relatively high pressures or weights which limits the strength of the structure as a whole. The flexibility of the pieces will result in a certain unpredictability in the behavior of the robot. The spaces between the joints of certain pieces also allows for a slight shift in the position of these pieces which also limits the reliability and sturdiness of the design.

The quality of the rubber wheels is also a source of limitation as there is sure to be a certain amount of slip between these wheels and the hardwood competition floors which, as seen in the labs, is a great cause of error in odometry and thus limits the performance of the robot.

The design of the EV3 brick used as the centerpiece of the robot is also a cause of limitation. The number of ports on the EV3 brick indeed limits the number of external components that can be used. The EV3 brick indeed only allows for the connection of 4 motors and 4 sensors.

3.2 Electromechanical Limitations

The motors are limited in the amount of torque they can produce. The motors used have a maximum running torque of 20 Ncm, and a stall torque of 40 Ncm. They also have a top rotational speed of 160-170 rpm depending on the battery level of the EV3 powering them. This range of max speeds can affect the calculations for the ball launcher since it will be powered by motors and an inconsistent rpm can cause a decrease in the accuracy of the ball thrower. Because of the inconsistencies, we must be conscious of the amount of power the battery will use throughout the round so that it is able to fire the ball with as high an rpm as possible. We will not have this issue with the motors used for driving the robot since we will never need to hit speeds this high.

The motors also have an error of one degree in the tachometer which can be accumulated to accrue a large error when the robot is traveling far distances. This is why odometry correction is mandatory.

The EV3 ultrasonic sensors are accurate with an error of 1 cm with a range of 1 to 255 cm. These limitations should not be a concern given what we are asking our ultrasonic sensors to do.

The limitation on the color sensor is that it samples at a rate of 1 KHz. This shouldn't be an issue but must be kept in mind when determining the speed of the robot. Another limitation lies within the nature of the color sensor where depending on the exterior lighting, the values might be inconsistent. We solved this issue by implementing a correction for when the light sensor misses a line on the field (see software document section 3.4).

The RJI2 cables will also limit our robot design since we must leave enough room in front of the ports so that the cables can be plugged in.

3.3 Electronic Limitations

The EV3 brick runs an ARM9 chip at 300MHz. Use of this brick during lab 5 demonstrated that a relatively large amount of threads running simultaneously was too much for the EV3 processor to handle and thus caused issues with the functioning of the robot. We will thus try to keep the number of threads to a minimum to ensure that the EV3 will be able to run the program with no issues. The EV3 brick also seems to have enough memory to suit the project needs as, during previous labs, no problems caused by lack of memory have been detected.

4 SOFTWARE AVAILABLE AND CAPABILITIES

Java: See Constraints Document, Section 4.

LeJOS: See Constraints Document, Section 4.

Eclipse: A commonly used Java IDE which has a plugin for the LeJOS API. This plugin also comes with an interface for easy control of communication with the EV3 brick. However, Eclipse often runs very slow and will sometimes crash a lot.

IntelliJ: A newer and increasingly more popular Java IDE which runs faster than Eclipse, with much more intelligent syntax correction. However, it does not have the LeJOS plugin and jars need to be imported manually. Communication with EV3 brick needs to be done through SSH as well.

Github: A web based version control system which uses git. The ability to work on different branches allows users to collaborate and work simultaneously on the same project. Conflicts may occur if two users work on the same lines of code, in which case higher level of experience in git is needed to solve efficiently.

Travis CI: An automated build system that is integrated into github. It follows a build file to clean, compile, and run any tests every time a commit is made onto the master branch to ensure the software is working at all times. Needs constant monitoring to ensure build script works accordingly.

LaTeX: A document preparation system to allows users to make clear and concise reports without having to worry about formatting. Usage of the tool requires knowledge of the LaTeX language and syntax.

Overleaf: A web based editing system for documents that use LaTeX. Allows users to collaborate on the same document and also includes a version control system. Documents can also be cloned as a git repository allowing edits to be made on a local development environment.

Gantt Project: Basic project management software that allows detailed Gantt Charts to be created. Unlike other modern applications, it is not a cloud based software making it hard to collaborate and share amongst team members.

Dropbox: A cloud storage platform that can be utilized to store all documentation and other relevant files.

Google Calendar: A Google application which allows the ability to create shared calendars to keep track of meetings and team member schedules.

Slack: A new open source communication and project management platform that allows custom integrations that range from multiple areas. Current integrations that are being used are:

- Busy Bot - Used to assign tasks and keep track of tasks that need to be finished.
- Google Calendar - Sends daily notifications for all events in shared Google calendar.
- Github - Sends notifications every time there is activity in the Github repository.
- Travis CI - Send notifications of every build result.
- Hour Stack - Allows team members to clock their hours simply by starting a timer when they start working and stopping it once they are done.
- Documents (Custom) - Custom made integration that prints a list of the links to all our LaTeX documents.

5 COMPATIBILITY

5.1 Software

The programming of the robot will be done in Java. The EV3 programming brick's default setup does not support Java so alternative firmware is provided. The Mindstorm environment run on top of Linux. The micro SD card contains a different version of Linux that has the LeJOS EV3 environment as a replacement user interface. Using the LeJOS EV3 environment in Eclipse, the robot is programmed using Java.
The programs developed in the weekly labs will be integrated into the robot in different ways. It was noticed during the lab that The code for the Wall Follower will not be completely compatible with the rules of the competition nor will it be efficient enough to be able to properly perform in the competition. It was indeed noticed during the lab that the wall follower method caused significant error to be accrued in the robot's odometry. For this reason it was decided that the wall follower method would be replaced with a different obstacle avoidance method for this project (see software document section 3.6 for details). The

code from the odometry lab will be used to allow the robot to keep track of its position on the field as it performed remarkably well. However, adjustments will have to be made before this code can be integrated into the current project as the code developed during the lab was calibrated to suit a smaller robot with a different build. The code developed for the Navigation lab will be used but adjusted significantly due to the nature of the competition being more complex than the one week lab. The code for localization used in lab 4 will be completely reworked due it not respecting the requirements of the current project which state that the robot must remain in its starting square for the duration of it's localization. The ball launcher code used in lab 5 will need to be adjusted to be compatible with the rules of the competition and the different robot build.

5.2 Hardware

Robot consists of parts only found in the EV3 Mindstorm kits each member has. Consequently, the different hardware components are highly compatible with each other. There are,however, certain constraints that limit the compatibility of the pieces (see section 3.1 of this document). Due to the new requirements of this project, most of the robot designs from previous labs will need to be modified before they can be integrated to the current project as they were not designed to perform the multiple tasks of this project.

6 REUSABILITY

See Section 5 of this document and Section 3 of the requirements document for the reusable hardware and software components of the labs.

7 STRUCTURES

7.1 Mechanical Structures

The decision was made to keep the hardware design of the vehicle simple as to limit the number of hours spent the physical build of the robot in order to allocate more resources perfecting the functioning of the robot. The main layout of the design is as follows.

Wheels: Two rubber wheels (provided in the Mindstorm kits) attached to two motors will be placed on either side of the vehicle to allow the robot to navigate. A small metal ball will also be placed at the back of the robot to ensure the rear of the robot does not drag on the floor when it moves. The back ball also makes the robot more stable which is crucial as we plan to mount the launching mechanism to the rear of the robot. From the previous labs we know that the robots tend to tilt forward when launching balls if the center of rotation of the launcher is far off the center of mass of the robot. To minimize this tilting motion we also plan to place a second metallic ball at the front of the robot.

Launcher: The design of the launcher will also be kept as basic as possible.The simplest design considered is described. We plan to simply have a tower-like structure built at the rear of the vehicle onto which we

will attach a motor-powered ball launcher at the top. The arm attached to the motors will hang at the back of the tower and will have a claw at its extremity on which the ball will rest. Other launcher designs are discussed in section 8.1.1 below.

7.2 Electrical Structures

Two possible design choices have been considered: one with two bricks and a second with a single EV3 brick. The 2 brick option would allow for a larger variety of robot designs and functions as it would allow for the connection of 4 additional motors and sensors. However, this option also presented flaws as our team had no experience in connecting the two bricks together. The 2 brick design would also greatly complicate the software of the project. On the other hand, the simpler, one brick option does not allow for as many connections but is far simpler alternative to the two brick design.

7.3 Software Structures

Refer to System Document, Section 2 for software architecture model.

User Interface Objects: The only interaction from the user will be through a user interface screen that is connected to an input object on the layer. This object will allow the user to select the round number before sending commands via WiFi to a server that will be hosted locally. The WiFi will then initialize the proper controller objects in the middle layer.

Controller Objects: Controller objects will be the bulk of the application. Objects that include a Localizer, Odometer, Ball Retriever, Defender, Navigator, and Launcher will work together to properly control the LEJOS API Objects in the bottom layer.

LEJOS API Objects: These objects each connect to a corresponding electrical hardware component. Each of the sensor objects will return an input from its corresponding hardware component to the controllers whenever they are requested. Each of the motor objects will output a physical movement to the corresponding hardware component when the command is sent from the controllers.

8 METHODOLOGIES

The different ideas brainstormed for the different hardware and software sections are described below. After running tests to assess each idea, a single idea per section will be selected and implemented into the final design.

8.1 Hardware

8.1.1 Launcher

- Tower-like structure supporting the throwing arm. Use of a single motor to rotate the throwing arm and launch the ball. Would free a motor for a different use.
- Tower-like structure supporting the throwing arm. Use of two motors to rotate the throwing arm
- Simple low centered firing mechanism powered by a single motor and attached to the base of the robot.
- Having the robot fire sideways instead of straight in front of it in order to confuse the defending team robot.
- Crossbow-like launching system where wheels would be attached onto two motors placed on the vehicle such that the wheels were at a 45 degree angle from the ground. The motors would then accelerate before a ball was placed in between both spinning wheels and launched forward.

8.1.2 Ball retrieval

- The idea was given of a ball container that would be placed on the side of the vehicle and would capture balls from the dispenser before feeding them to the throwing arm. This would allow for multiple balls to be carried at once.
- Simply lifting the throwing arm to the height of the dispenser and collecting one ball at a time

8.1.3 Sensors

- Have 3 ultrasonic sensors and 1 light sensor. The 3 ultrasonic sensors would be placed at the front of the robot facing North, East and West which would allow the robot a complete view of its surroundings and a more effective obstacle avoidance. The light sensor would be placed at the bottom of the robot and would be used for odometry correction.
- Having 1 stationary ultrasonic sensor pointing towards the front of the robot while a second ultrasonic sensor was mounted on a motor. 2 light sensors would then be placed at the bottom of the robot.
- Having 2 ultrasonic sensors attached onto a front motor that would be able to rotate and give the robot a full view of its surroundings without sacrificing the use of a second light sensor. 2 light sensors would be placed underneath the robot on either side of its body and would be used for odometry correction.
- Same as last idea but having the 2 ultrasonic sensors be stationary instead of mounted on a motor.

8.2 Software

8.2.1 Navigation, Odometry and Odometry Correction

The Robot will always navigate in such a way to cross the gridlines perpendicularly in order to increase odometry accuracy. See software document section 3.5 for the complete navigation procedure. Odometry will be calculated using the tachometer in the same way it was used for the labs. The odometry correction procedure will differ depending on the hardware design chosen in the previous section. The different ideas for odometry correction are discussed below:

- Using 2 light sensors,when he robot comes across a line perpendicular to its velocity, once the first light sensor hits the line the corresponding wheel stops and waits for the next light sensor to reach the line which allows for correction of the robots heading. The odometry is then corrected to the exact odometry of that line.
- Using a single light sensor the robot would correct its X and Y coordinates every time it hit a line but it's heading could not be corrected. A second disadvantage to this method was also identified: If the light sensor did not detect the line on the first pass it would have no way of knowing and would correct its odometry to the wrong coordinates.

8.2.2 Localization

Because the the constraints of the competition, a single possible localization method was considered. The robot will rotate counterclockwise in its starting square and use the data from its ultrasonic sensor to determine the 2 closest points which will then be registered as the X and Y coordinates respectively. The robot will also store it's odometer heading value at these two points in order to correct its heading once the procedure is complete (see software document section 3.7).

8.2.3 Obstacle Avoidance

- Using 3 ultrasonic sensors placed as mentioned in section 8.1.3 above the robot would navigate until the front sensor detected an object at which point it would turn either 90° or -90° depending on its destination and keep advancing till ultrasonic sensor on it's corresponding side no longer sensed an obstacle.
- Using only 2 sensors mounted on a motor the robot would follow the same procedure as described above but would rotate the motor to ensure that a sensor is pointed on the side on which the obstacle is located.
- Using a map of the field the robot would indicate the placement of the detected obstacles and avoid the squares in which obstacles are present.
- Using two fixed sensors the robot would be able to detect an obstacle on one of its sides without have to turn to that side. If the robot needed to turn on the other side, however, the front sensor would be used to detect obstacles. The idea with two fixed sensors is that the robot would be able

to map part of the field as it traveled using the side sensor and know in advance where the obstacles were placed.

- Using only 1 ultrasonic sensor, whenever the robot reaches the middle of a tile, it will stop and evaluate its next move based on the navigation algorithm. It will turn to face its next move and then swivel so the front sensor can check the front tile for obstacles. If an obstacle is detected by the front sensor, the robot will find the tile around it with the next priority and travel there. This is a very slow method on the first run as the robot must scan every square but, combined with the mapping described above, it becomes very fast on subsequent runs as it no longer needs to scan squares for obstacles.

8.2.4 Boundary Avoidance

Boundary avoidance will use a mapping of the field that is initialized accordingly based on user inputs. It will use odometer values to determine its position within the boundaries. If a boundary is approaching, it will implement a wall follower method except it will replace the ultrasonic sensor readings with distances calculated from the odometer. Just like the obstacle avoidance, this will be implemented into the navigator object and interrupt the navigation thread anytime a boundary is detected.

8.2.5 Retrieving Balls

In order to retrieve balls, it will use the mapping of the field initialized at the beginning of the program to know the exact position of where the balls are located. Upon retrieving this position, it will then use the navigator class to move the robot to the tile next to the desired location to retrieve the ball. The actual ball retrieval process will very depending on the chosen hardware design:

- For the tower and motor designs the robot will start by lowering its arm from its traveling position (on top of the robot). Once the arm is down the robot will align itself using the light sensor(s), back up a set distance and raise its arm to the level of the dispenser and beep to collect a ball.
- For the ball container idea the robot would follow the same process except it would have to back up further and turn to the side the container is located on before beeping and retrieving the balls.

8.2.6 Scoring

In order to score, the robot will begin by using the navigator object to move into a position that is optimal to shoot from. It will line up to the the line perpendicular from the target and then the line parallel to the target to correct its odometry similar to ball retrieval. Then, depending again on the chosen hardware model, the robot will fire the ball. Different procedures for this follows.

- With the tower and launching arm design, the ball launching motors will rotate the arm at a specific rotational velocity and to a specific release angle depending on the robots current position.
- for the crossbow design, the motors would have to accelerate before a ball was fed to the shooting mechanism and fired.

9 TOOLS

See System Document, sections 3 and 4 (Hardware/Software available).

10 GLOSSARY OF TERMS

- RJ12 cable: Connector cables used to connect the main EV3 brick to other electrical components.
- Ball launcher: The mechanism used to launch the ball
- Mindstorm EV3 kit: This is the hardware kit containing the majority of the hardware components to be used in this project.
- API: The Application program interface is a set of routines, protocols, and tools for building software applications
- Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network
- Slack integration: an addition to a specific slack channel that allows for additional functionalities by the users of the channel
- Slack channel: A channel in the slack application through which users communicate. Multiple slack channels can be created at once.
- EV3 programming brick: The main part of the mindstorm kit. This is the component of the robot on which the software will run.
- Lever: Part of the robot used to launch the ball. Also referred to as the launching arm.