SYSTEM DOCUMENT

Project: BOB: Bob's Our Builder
Task: To collect blocks and build a tower.
Date: November 24, 2016
Author(s): Adham El-Khouly, Camilo Garcia, Jenny Zhao, Jackson Wang, Mathieu Lapointe, Juliette Regimbal
Document Version Number: 4.0
Revision History:
V1.0: Created Section 1.0-9.0
V2.0: Section 2.3 updated to accommodate newest Design decision of using 1 EV3; (Adham El-Khouly)
V3.0: Section 5.0 "Reusability" updated to include software reusability; (Juliette Regimbal)
V4.0 Final Version: Finalized Project name to "*BOB: Bob's Our Builder*"; Replaced System Model Diagram with new version; Expanded Section 2.1 to include "*EV3 Brick, Large Regulated Motor, Medium Regulated Motor*"; Updated Section 6.0; Created sections 6.1, 6.2; Created "*Glossary of Terms*". (Juliette Regimbal, Jenny Zhao)

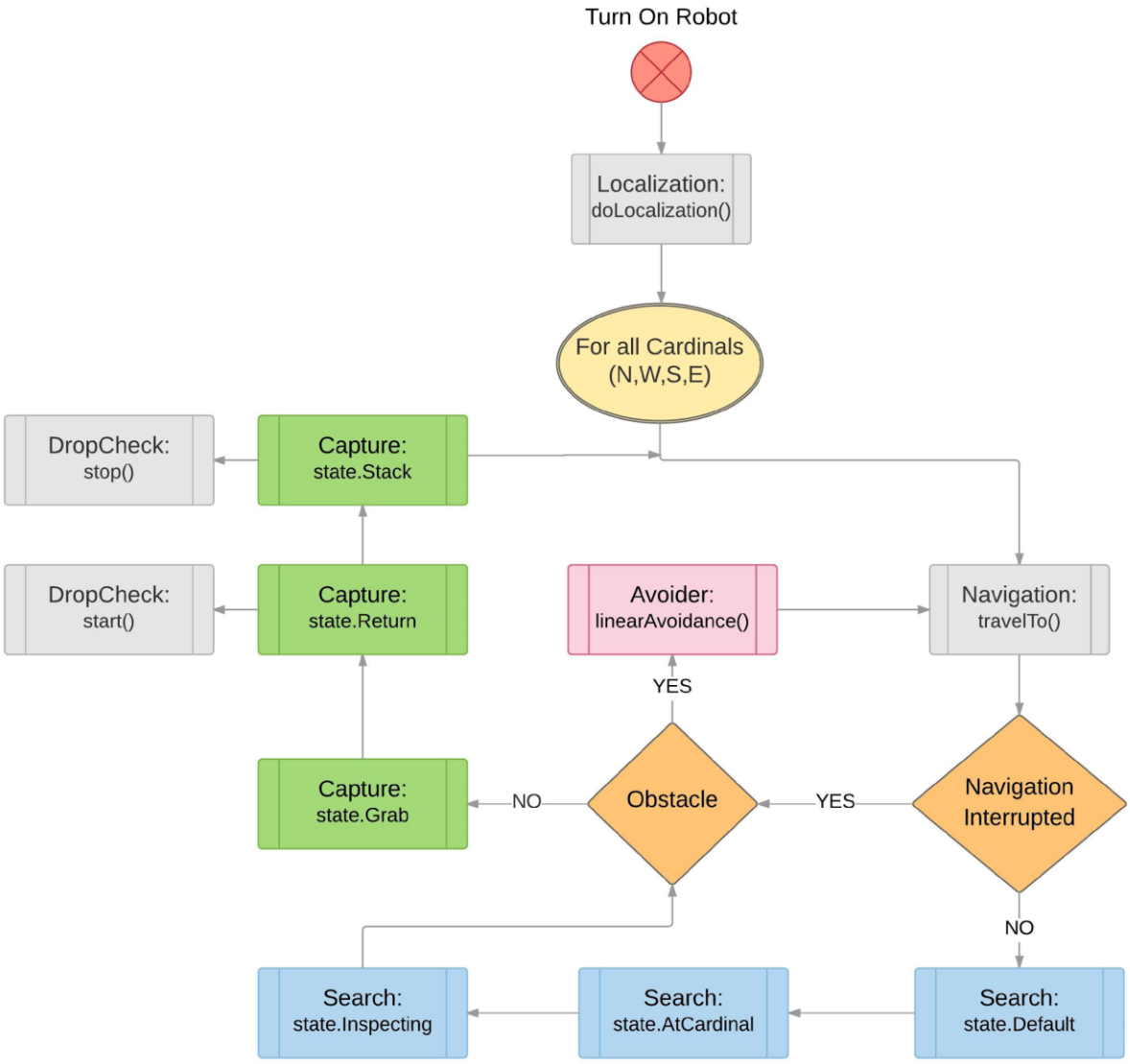
Table of Contents

1.0 SYSTEM MODEL

As described in the Requirements Document section 2.0 Purpose, the final robot can be assigned one of two modes: Builder or Garbage Collector. While both require common basic robotic functionalities, some processes will be unique to each operation mode. System functionalities such as localization, navigation, object detection, object delivery and obstacle avoidance are shared among both roles. As such, they represent the core of the efficiency of the robot. This classes will undergo the most intensive testing of the project, for we require them to be the *ne plus ultra* of our final product to better predict the behavior of more advanced functions specific to each mode. The most prominent example of such specific task is seen in the builder role. It must employ an extra class which will allow the robot to stack blocks in an efficient and prompt way.

The first revision of our model works under the main idea of a state automata. Each class has a series of predefined and constant states, only one active per class at any given moment of the execution. This states are toggled by other classes and by external inputs, such as a distance measurement, a color identification or a wheel rotation.

The following diagrams provide a visual overview of the aforementioned model:


Turn On Robot
Localization:
doLocalization()
For all Cardinals
(N,W,S,E)
DropCheck:
stop()
Capture:
state.Stack
DropCheck:
start()
Capture:
state.Return
Avoider:
linearAvoidance()
Navigation:
travelTo()
YES
Capture:
state.Grab
NO
Obstacle
YES
Navigation
Interrupted
NO
Search:
state.Inspecting
Search:
state.AtCardinal
Search:
state.Default

2.0 HARDWARE AVAILABILITY & CAPABILITY

2.1 Lego Components and Their Mechanical Capabilities

Using the Lego kit pieces does indeed pose many limitations as previously mentioned in the Constraints document, in section 3.0, especially due to their size. The rest of the components in the kits each have their limitations as well.

- *EV3 Brick:*
  - 4 input ports, 4 output ports
  - 300MHz ARM9 Processor
  - Connectivity through USB, Bluetooth, Wifi

- *Ultrasonic Sensor:* The EV3 ultrasonic sensor emits sounds waves and detects their echoes once they bounce off surrounding objects.
  - Should measure distance from 1cm to 255cm with a +/- 1cm error as mentioned on the device specifications. However, it reads maximum distance if it is too close to an object. During tests it also proved to have an error of +/- 3cm.
  - Having other robots with ultrasonic sensors on the field at the same time also interferes with the sensor readings.

- *Color Sensor:* The EV3 color sensor can function as a color sensor or as a light sensor detecting different light intensities. It has a sampling rate of 1kHz and is able to distinguish between 8 different colors.

  - The color sensor was tested in the same room every time, therefore ambient light could potentially interfere with its readings.

- *Touch Sensor:* The EV3 Touch sensor detects when its front button is pressed or released

  - The touch sensor is binary, in the sense that it has two states, either pressed or released. This limits its usage to a few tasks, for example, obstacle avoidance. It would be impossible to use the touch sensor to check for material firmness, for example.

- *EV3LargeRegulatedMotor:* basic control methods used: *forward, backward, stop.*

- Maximum rotation speed of 165 RPM
- Stalled Torque: 43 N.cm

- *EV3LargeRegulatedMotor:* basic control methods used: *forward, backward, stop.*
  - Maximum rotation speed of 260 RPM
  - Stalled Torque: 15 N.cm

2.2 Electromechanical Limitations

Electromechanical limitations can be summarised to the delay between processing the code and doing the action required.

2.3 Processor Constraints

The EV3 brick utilises an ARM9 32-bit processor. The use of one brick as only 4 input ports and 4 output ports are needed was deemed sufficient. The mechanical design was simplified significantly, which allowed for the software part of the design to also be very simple. Therefore, the use of one brick generated enough processing power to carry out the required tasks.

3.0 SOFTWARE AVAILABILITY & CAPABILITY

As specified in the Constraints document, section 4.2 Software constraints – Programming language, we are constrained to the leJOS java virtual machine environment ported to the EV3 block in 2013, but we had other options in terms of programming frameworks to use, such as NXTG and RoboLab.

The last two projects (NXTG and RoboLab) have a more educational approach, complemented by a graphical programming approach seen in high-school academic programming languages such as scratch. In consequence, the coding structure proposed for our project would suffer from low responsiveness and large overhead in terms of high level encapsulation and file size. The robot processes require a far more efficient speed of execution and less object abstraction. We have the programming capabilities to produce efficient, modular code without a graphical drag-and-drop approach. As such, we will focus on the broad range of

capabilities that the team has at its disposal, most of which are intrinsic in robotics, in the Java language. Most notably:

- Object Oriented programming: To maintain and structure in a logical and concise way all the functionalities of our project, we require a modular language. OO programming is the natural answer to this problem.
- Pre-emptive Threads: most of the processes during our robots final run will run in parallel. As opposed to cooperative threads, the pre-emptive approach to multithreading will force us to consider the bigger overhead of the process and the priority given to each thread. To prevent unexpected behaviors due to this methodology, each thread should run constantly, but only perform an action based on the change in state of a Boolean or enum variable.
- Synchronization: To avoid race conditions and data corruption, all variables which bestow from samplers or which are used across classes will implement mutual exclusion. This will be achieved by lock objects which will synchronize data access and modification.
- Exceptions: especially in the debugging stage, the team will take advantage of the capability to throw exceptions to facilitate the process of error handling and prevention.

(See the Software Document for more information)

REFERENCES:

https://docs.oracle.com/javase/7/docs/api/
https://shop.lego.com/en-CA/EV3-Color-Sensor-45506
https://shop.lego.com/en-US/EV3-Ultrasonic-Sensor-45504
https://shop.lego.com/en-US/EV3-Intelligent-Brick-45500
https://www.arm.com/products/processors/classic/arm9/index.php

4.0 COMPATIBILITY

4.1 Software Compatibility

- EV3 brick must be programmed from a computer, with LEGO MINDSTORM EV3, or with other programming languages such as JAVA, with LeJOS package plugin
- LeJOS is compatible with JavaSE 1.7 or lower

- EV3 operates Mindstorm system version Beta 9.0.1 or lower

4.2 Hardware Compatibility

- EV3 is backward compatible with NXT sensors, motors
- EV3 is compatible with LEGO building elements

4.3 Communication Compatibility

- Human can communicate with EV3 through USB connection, wifi, or Bluetooth
- EV3 to EV3 communication is possible through LeJOS

5.0 REUSABILITY

- LEGO building elements can be disassembled from the robot and re-used in other building projects
- Classes adding another level of abstraction to sensors can be used for other applications.
    - Ultrasonic sensor, Color sensor, Light intensity sensor, and their associated methods.
- The state system implemented in the robot can be expanded upon or revised for other projects using LeJOS and an EV3.

6.0 STRUCTURES

For details of both Mechanical and Software Structures, please refer to the Hardware Documentation and Software Documentation.

6.1 Hardware

6.1.1 Robot Design 1.0: Prototype

The robot will consist of two EV3 bricks, two large motors, and a grabbing arm. Initially we thought of designing our robot with a forklift, but the idea was soon deemed unreliable since it would be very difficult using a forklift to stack blocks onto each other. The idea of using two EV3 blocks and two motors was approved by the team because using motors would make the system unreliable and harder to debug.

6.1.2 Robot Design 2.0, and 2.1

The robot is now consisting of one EV3 brick, 2 Motors for moving the robot, 1 for the grabbing arm. As for sensors, we decided to use 1 Ultrasonic Sensor to detect obstacles, and 2 light sensors: 1 to detect grid lines, 1 for object differentiation. At this stage, a prototype of the dual brick chassis and grabbing arm has been built.

6.1.3 Robot Design 3.0, 3.1, 3.2 and 3.4 (BOB)

The design of using one EV3 brick is finalized. The decision is made based on the choice of number of sensors we are using, the chosen sensors are: 1 Ultrasonic Sensor to detect obstacles; 2 light sensors, one to detect grid lines, one for object differentiation. Since 3 sensors are within the limit of 4 input ports provided by 1 EV3 brick, we finalized our decision of using 1 EV3 brick. Other parts include 2 Motors for moving the robot, 1 for the grabbing arm, and 1 more for operating the claw.

Another important change is that the forklift design is retrieved and replaced the grabbing arm. It was modified several times during the process, mostly to increase the height to accommodate block stacking. It's operated by a pulley system that utilizes a medium motor to roll the cord.

6.2 Software

6.2.1 Javadoc 1.0

In the first meeting, the team decided to use state machines for the project. The overall controller will be made of multiple states, in each state the robot will perform a certain set of instructions. Also, the team decided to never stop the current thread, this is because in Java there's no clean, quick or reliable way to stop a thread. Stopping a thread usually causes unexpected errors and system corruptions. The use of descriptive names for variables and helper methods was also agreed upon. See Javadoc 1.0 for detail.

6.2.2 Javadoc 2.0

Control for the LCDInfo helper class was added and documented. The sensor classes added - Ultrasonic Sensor, Color Sensor, and Light Intensity Sensor - also were given descriptions in the updated Javadoc. The demo states were documented, as well as the main function and demo state selection function. Avoider and ThreadEnder were added. Individual test functions were given descriptions of their purpose. Descriptions and parameters received minor updates to reflect the changes made between versions 1 and 2 where necessary.

6.2.3 Javadoc 3.0

Major changes to Avoider and Capture, resulting in them being functional. Forklift class added to control the robot's forklift. A DropCheck class added that runs in a separate thread during capture to notify the robot if the block is dropped. Then the robot begins searching for the block to recover it, and if that is successful resumes capture operations. Tests for avoidance, returning to a starting corner, and the forklift were added and documented. Minor changes to the javadocs were made to make them more descriptive.

7.0 METHODOLOGIES

This will be done as a group in the meetings. The basis for the algorithms (using state machine) as well as the initial hardware design approach are presented herein under section 6.0. For initial ideas of hardware design, pseudo code and detailed design iterations, please see the Software Documentation and Hardware Documentation.

8.0 TOOLS

Please refer to the following documents for the most detailed, relevant Tools information:
"Section 3.0: Software Availability & Capability"
"Constraints Document: Section 3.0 Hardware Constraints"

9.0 GLOSSARY OF TERMS

EV3 - NXT

The EV3 and the NXT are both robotics kits. The EV3 is the newer version.

Light Sensor

A light sensor is an electronic unit that can measure the light intensity.

Ultrasonic Sensor

An ultrasonic sensor is an electronic unit that sends out sound waves and measures the time it takes for those waves to bounce back. By knowing the speed of sound the sensor is able to calculate the distance to those objects.

Pulley System

A system that relies on a string that rotates around a pulley in order to move an object in a certain way, in our case, vertically for the forklift, which the claw is attached to.

Cord

A piece of fishing wire used within the pulley system.

Thread

In computer science, a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler.